USB-2000 Series

USB 2.0 Full-Speed High Performance DAQ module

User's Manual

Revision History

Revision	Date	Description of Change
1.14	Oct 29,2015	1. Modify some of the description
1.13	Aug 12,2015	2. Adding USB-2026 Module Information 3. Adding AO API for output function
1.12	Jan 13, 2015	1. Modifying the wiring diagram for USB-2051, USB-2055 and USB-2060
1.11	Aug 28, 2014	1. Adding DO API for output inversing function 2. Modifying DI API for reading counter value from 16-bit to 32-bit
1.10	Aug 5, 2014	1. Adding firmware upgrade instruction in Appendix C
1.09	May 8, 2014	1. Adding USB-2045 Module Information 2. Adding USB-2051 Module Information 3. Adding USB-2055 Module Information 4. Adding USB-2060 Module Information 5. Adding DI API
1.08	Jan 29, 2013	1. Adding System API – SetAutoResetWDT
1.07	Sep 20, 2012	1. Adding DO API – DO_WriteValue
1.06	Jul 6,2012	1. Adding DO API 2. Adding information for USB-2064
1.05	Jun 21, 2012	1. Modifying analog output type code
1.04	Apr 23, 2012	1. Changing the color of LED indicators
1.03	Dec 29, 2011	1. Adding information for USB-2084 2. Adding PI API 3. Adding PI related error codes
1.02	Dec 20, 2011	Adding ERR_USBDEV_ERROR_WRITEFILE error code
1.01	Dec 15, 2011	Modify specification of USB-2019
1.00	Oct 31, 2011	First revision released

Preface

Warranty

All products manufactured by ICP DAS are under warranty regarding defective materials for a period of one year from the date of delivery to the original purchaser.

Warning

ICP DAS assumes no liability for damages resulting from the use of this product. ICP DAS reserves the right to change this manual at any time without notice. The information furnished by ICP DAS is believed to be accurate and reliable. However, no responsibility is assumed by ICP DAS for its use, or for any infringements of patents or other rights of third parties resulting from its use.

Copyright

Copyright © 2011 by ICP DAS CO., LTD. All rights are reserved.

Trademark

The names used for identification only may be registered trademarks of their respective companies.

Content

Revision History i
Preface ii
Content iii
1 Introduction 1
1.1 Overview 1
1.2 Feature 1
1.3 Applications 2
1.4 Specifications 2
1.4.1 General 2
1.4.2 USB-2019 3
1.4.3 USB-2026 4
1.4.4 USB-2045 7
1.4.5 USB-2051 8
1.4.6 USB-2055 9
1.4.7 USB-2060 11
1.4.8 USB-2064 13
1.4.9 USB-2084 14
1.5 Product Check List 15
2 Hardware Information 16
2.1 Module Overview 16
2.1.1 USB-2019 17
2.1.2 USB-2026 & USB-2045 & USB-2051 & USB-2055 &USB-2060 18
2.1.3 USB-2064 18
2.1.4 USB-2084 19
2.1.5 CA-USB15 19
2.2 Connector Pin Assignment 20
2.2.1 USB-2019 20
2.2.2 USB-2026 20
2.2.3 USB-2045 21
2.2.4 USB-2051 22
2.2.5 USB-2055 22
2.2.6 USB-2060 23
2.2.7 USB-2064 23
2.2.8 USB-2084 24

2.3 Wiring 25
2.3.1 USB-2019 25
2.3.2 USB-2026 25
2.3.3 USB-2045 26
2.3.4 USB-2051 27
2.3.5 USB-2055 27
2.3.6 USB-2060 28
2.3.7 USB-2064 29
2.3.8 USB-2084 29
2.4 Hardware Configuration 30
2.4.1 Board ID 30
2.4.2 Firmware Update 30
2.4.3 USB-2019 31
2.4.4 USB-2026 31
2.4.5 USB-2045 32
2.4.6 USB-2051 32
2.4.7 USB-2055 32
2.4.8 USB-2060 33
2.4.9 USB-2064 33
2.4.10 USB-2084 34
2.5 LED Indicators 35
2.5.1 Normal Operation 35
2.5.2 Firmware update 35
3 Installation 36
3.1 Hardware 36
3.1.1 Connecting to ICP DAS USB series I/O module 36
3.2 Software 36
3.2.1 Utility 37
3.2.2 ICP DAS USB I/O Software Integration 47
3.2.3 Samples 49
4 Operation 50
4.1 Hardware structure 50
4.2 Software structure 50
5 ICP DAS USB Class Members 53
5.1 Table of Constructors 53
5.2 Table of Static Methods 53

5.3 Table of Public Methods 53
5.3.1 System 53
5.3.2 Device 54
5.3.3 Digital Input 54
5.3.4 Digital Output 55
5.3.5 Analog Input 55
5.3.6 Analog Output 57
5.3.7 Pulse Input 57
5.3.8 Other 58
5.4 Constructors 59
5.4.1 ICPDAS_USBIO 59
5.5 Static Methods 60
5.5.1 ListDevice 60
5.5.2 ScanDevice 61
5.6 Public Methods 62
5.6.1 System 62
5.6.2 Device 68
5.6.3 Digital Input 89
5.6.4 Digital Output 104
5.6.5 Analog Input 129
5.6.6 Analog Output 167
5.6.7 Pulse Input 232
6 Troubleshooting 274
Appendix A 275
A.1 Analog Input Type Code 275
A.2 Analog Output Type Code 276
A.3 Pulse Input Type Code 276
A.4 Channel Status 276
Appendix B 277
B.1 Error Codes 277
Appendix C 280
C.1 Steps of updating firmware for USB I/O module 280

1 Introduction

1.1 Overview

The ICP DAS USB series I/O modules are highly flexible solution for data acquisition. It provides easy USB plug-and-play operation and equips accurate measurement for all kinds of applications of automations. Compared with the traditional PC-based cards like PCI, PC/104 and ISA cards, users can achieve data acquisition easier and quicker via ICP DAS USB series I/O modules. Besides, through ICP DAS USB I/O utility, users can configure and test modules directly and easily without any coding. The friendly API library is also provided for users to develop own USB application.

1.2 Feature

- Maximum 10KS/s sampling rate
- Wide operating temperature range
- RoHS compliant
- USB 2.0 Full-Speed compliant
- No external power supply (Powered by USB)
- Plug-and-Play without driver installation
- Lockable USB cable
- Support firmware update via USB
- Utility tool for module configuration and I/O testing easily and quickly
- PWR/RUN/ERR LED indicator
- Built-in dual watchdog (hardware/software)
- Providing API Library (VC/VB/BCB/.NET)
- Module supported for Win2000/XP and Win7 (32/64 bit)

1.3 Applications

- Building automation
- Factory automation
- Machine automation
- Data acquisition and control
- Environment monitor
- Laboratory equipment and research

1.4 Specifications

1.4.1 General

Communication	
Interface	USB 2.0 Full-Speed
Watchdog	1 Hardware watchdog (1.6 second) 1 Software watchdog (Programmable)
LED Indicators / Display	
System LED Indicators	3 LED as Power, Run and Error
I/O LED Indicators	1 LED / channel as I/O status for Digital and Pulse I/O
EMC	
ESD (IEC 61000-4-2)	4 kV contact for each terminal 8 kV air for random point
EFT (IEC 61000-4-4)	0.5kV for USB cable 0.5kV for I/O terminal
Environment	
Operating Temperature Range	-25 ~ +75℃
Storage Temperature Range	-40 ~ +85℃
Humidity	10 ~ 95% RH, non-condensing

1.4.2 USB-2019

The USB-2019 is an 8-channel universal analog input module. It supports the over-voltage protection of up to 240Vrms. In addition, it has voltage and current input types. It also widely supports thermocouple devices with J, K, T, E, R, S, B, N, C, L, M and $L_{DIN43710}$ types. Moreover, it provides extremely accurate thermocouple measurement and automatically cold-junction compensation for each channel. Finally, it features open wire detection for thermocouple and 4 ~ 20 mA inputs for each channel.

Analog Input		
Channels		8 differential
Input Type	Voltage	±15 mV, ±50 mV, ±100 mV, ±150 mV, ±500 mV, ±1 V, ±2.5 V, ±5 V, ±10 V
	Current	±20 mA, 0 ~ +20 mA, +4 ~ +20 mA (Note : An external resistor is required)
	Thermocouple	J, K, T, E, R, S, B, N, C, L, M and $L_{DIN43710}$
Resolution		16 bit
Accuracy		±0.1% FSR
Sampling Rate		10 Hz (Total)
Zero Drift		±20 μV/℃
Span Drift		±25 ppm/℃
Common Mode Rejection		86 dB
Normal Mode Rejection		100 dB
Input Impedance	Voltage	> 400 kΩ
	Current	125Ω (External resistor is required)
Intra-Module Isolation, Field-to-Logic		3000 V_{DC}
Overvoltage protection		240 V_{rms}
Individual Channel Configuration		Yes
Open Wire Detection		Yes (Software programmable)
Power		
Power Consumption		1.45W maximum
Mechanical		
Dimensions (W×L×H)		33mm × 119mm × 107mm

1.4.3 USB-2026

The USB-2026 is a multifunction module that includes 5 analog input channels, 2 analog output channels, 2 digital input channels, 2 digital output channels and compatibles with USB 2.0 full-speed. It equips small size, portable, USB bus powered, various input type features to help user build up own project easily and quickly. The USB-2026 provides a programmable input range on all analog input (+/-150 mV, +/-500 mV, +/-1 V, +/-5 V, +/-10 V, +/-20 mA or 0 ~ +20 mA), analog outputs are 12-bit at +/-5 V, +/-10 V, 0 ~ +20 mA or +4 ~ +20 mA. Each analog input can be configured for an individual range and provides a high overvoltage protection of 240 Vrms.

Multifunction		
Analog Input		
Channels		5 Differential
Range		Voltage: ±1 V, ±2.5 V, ±5 V, ±10 V
		Current: ±20 mA, 0 ~ 20 mA, 4 ~ 20 mA
Resolution	Normal Mode	14-bit
	Fast Mode	12-bit
Sampling Rate	Normal Mode	10 Hz total
	Fast Mode	200 Hz total
Accuracy	Normal Mode	±0.1 % FSR
	Fast Mode	±0.5 % FSR
Zero Drift		±20 μV/°C
Span Drift		±25 ppm/°C
Input Impedance	Voltage	20 MΩ
	Current	135 Ω
Overvoltage Protection	Voltage	120 VDC
	Current	N/A
Overcurrent Protection	Voltage	N/A
	Current	Yes, 50 mA at 110 VDC
Analog Output		
Channels		2
Range (Jumper Selectable)		Voltage: +0~+5 VDC, +0~+10 VDC, ±5 VDC, ±10 VDC,

		Current: +0~+20 mA, +4~+20 mA
Resolution		12-bit
Accuracy		±0.1% of FSR
Output Capacity Voltage Output Capability		10 V @ 20 mA
Open Wire Detection		Yes, for 4 ~ 20 mA only
Power On Value		Yes
Safety Value		Yes
Digital Input		
Channel		2
Type		Dry Contact (Source)
On Voltage Level		Close to GND
Off Voltage Level		Open
Input Impedance		10 KΩ, 0.5 W
Counters	Channels	2
	Max. Count	4,294,967,285 (32-bit)
	Max. Input Frequency	100 Hz
	Min. Plus Width	10 ms
Overvoltage Protection		±57 VDC
Digital Output		
Channels		2
Type		Isolation Open Collector (Sink)
Max. Load Current		700mA/Channel
Load Voltage		+3.5 ~ 50 VDC
Overvoltage Protection		±60 VDC
Overload Protection		1.4 A (with short-circuit protection)
Short Circuit Protection		Yes
Power-on Value		Yes, Programmable
Safe Value		Yes, Programmable
Communication		
Interface		USB 2.0 Full-Speed
Watchdog		1 Hardware watchdog (1.6 second)
		1 Software watchdog (Programmable)
LED Indicators		
System LED Indicators		3 LED as Power, Run and Error

EMS Protection	
ESD (IEC 61000-4-2)	4 kV contact for each terminal
	8 kV air for random point
Mechanical	
Dimensions	33mm × 78mm × 107mm
Environment	
Operating Temperature	-25 ~ +75℃
Storage Temperature	-40 ~ +85℃
Humidity	10 ~ 95% RH, non-condensing

1.4.4 USB-2045

The USB-2045 is a full-speed USB device with 16 digital output channels module. The USB-2045 supports source type output and equips with short circuit protection. There are 16 LED indicators that can be used to monitor the status of the digital output channels. The 4 kV ESD protection, 4 kV EFT protection, 3 kV surge protection for power input and 3750 VDC Intra-module isolation are standard.

Digital Output	
Channels	16
Type	Open Collector, Sink (NPN)
Load Voltage	+3.5~+50VDC
Max. Load Current	650 mA/Channel
Overvoltage Protection	60 VDC
Overload Protection	1.4A (with short-circuit protection)
Power-on Value	Yes
Safe Value	Yes
Power	
Power Consumption	1.0 W max.
Mechanical	
Dimensions (W×L×H)	72 mm x 123 mm x 35 mm

1.4.5 USB-2051

The USB-2051 is a full-speed USB device with 16 digital input channels module. The USB-2051 offers 16 channels for digital input, catering for both dry and wet contact, with an effective distance for dry contact of up to 500 meters. All channels not only feature photocouple isolation, but can also be used as 16-bit counters. The USB-2051 has 16 LED indicators that can be used to monitor the status of the digital input channels. 4 kV ESD protection and 3750 VDC intra-module isolation are standard.

Digital Input		
Channels		16
Type	Dry Contact	Source
	Wet Contact	Sink/Source
On Voltage Level	Dry Contact	Close to GND
	Wet Contact	+10 VDC ~ +50 VDC
Off Voltage Level	Dry Contact	Open
	Wet Contact	+4 VDC Max.
Effective Distance For Dry Contact		500 meters Max.
Input Impedance		10 KΩ
Overvoltage Protection		70 VDC
Counter	Max. Count	65535 (16-bit)
	Max. Input Frequency	500 Hz
	Min. Pulse Width	1 ms
Power		
Power Consumption		1.03 W max.
Mechanical		
Dimensions (W×L×H)		72 mm x 123 mm x 35 mm

1.4.6 USB-2055

The USB-2055 is a full-speed USB device with 8 digital input and digital output channels module. The USB-2055 offers 8 isolated channels for digital input and 8 isolated channels for digital output. Either sink-type or source-type digital input can be selected via wire connections. All digital input channels are also able to be used as 16-bit counters. The USB-2055 supports source-type output with short circuit protection. There are options to enable both power-on and safety values. The USB-2055 has 16 LED indicators that can be used to monitor the status of the digital input and digital output channels. 4 kV ESD protection and 3750 VDC intra-module isolation are standard.

Digital Input		
Channels		8
Type	Dry Contact	Source
	Wet Contact	Sink/Source
On Voltage Level	Dry Contact	Close to GND
	Wet Contact	+10 VDC ~ +50 VDC
Off Voltage Level	Dry Contact	Open
	Wet Contact	+4 VDC Max.
Effective Distance For Dry Contact		500 meters Max.
Input Impedance		10 KΩ
Overvoltage Protection		70 VDC
Counter	Max. Count	65535 (16-bit)
	Max. Input Frequency	500 Hz
	Min. Pulse Width	1 ms
Digital Output		
Channels		8
Type		Open Collector, Sink (NPN)
Load Voltage		+3.5~+50VDC
Max. Load Current		650 mA/Channel
Overvoltage Protection		60 VDC
Overload Protection		1.4A (with short-circuit protection)

Power-on Value	Yes
Safe Value	Yes
Power	
Power Consumption	1.2 W max.
Mechanical	
Dimensions (W×L×H)	33mm × 87mm × 107mm

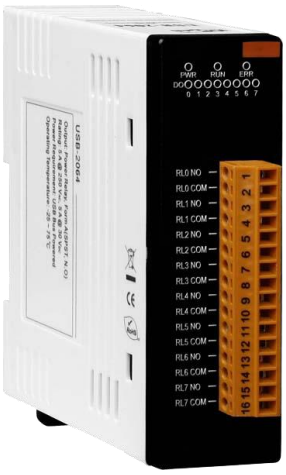
1.4.7 USB-2060

The USB-2060 is a full-speed USB device with 8 digital input and digital output channels module. The USB-2060 provides 6 digital input channels, 6 Form A signal relay output channels. All digital input channels can be used as 16-bit counters. In addition, the digital input channels can be selected either as sink or source type via wire connections. The USB-2060 also provides 12 LED indicators that can be used to monitor the status of the digital input and relay output. There are also options for configuring power-on and safe values. 4 kV ESD protection and 3750 VDC intra-module isolation are also provided to enhance noise protection capabilities in industrial environments.

Digital Input		
Channels		6
Type	Dry Contact	Source
	Wet Contact	Sink/Source
On Voltage Level	Dry Contact	Close to GND
	Wet Contact	+10 VDC ~ +50 VDC
Off Voltage Level	Dry Contact	Open
	Wet Contact	+4 VDC Max.
Effective Distance For Dry Contact		500 meters Max.
Input Impedance		10 KΩ
Overvoltage Protection		70 VDC
Counter	Max. Count	65535 (16-bit)
	Max. Input Frequency	500 Hz
	Min. Pulse Width	1 ms
Relay Output		
Channels		6
Output Type		Form A (SPST-NO)
Contact Rating (Resistive Load)	DC	5A 30V_{DC}
	AC	5A 250V_{AC} (47~63Hz)
Operate Time		10ms max.
Release Time		5ms max.

Insulation Resistance		1,000 MΩ at $500V_{DC}$
Dielectric Strength	Between Open Contact	$1000V_{AC}$ (1 min.)
	Between Coil and Contact	$3000V_{AC}$ (1 min.)
Endurance	Mechanical	2×10^7 ops
	Electrical	1×10^5 ops
Power-on Value		Yes
Safe Value		Yes
Power		
Power Consumption		1.3 W max.
Mechanical		
Dimensions (W×L×H)		33mm × 87mm × 107mm

1.4.8 USB-2064



The USB-2064 is an 8-channel power relay output module with the USB interface. It provides a maximum 5A driving load current for variety applications. In addition, This module also provides the safety functionality to secure devices in the field. Besides, it also supports power-on configuration to customize initial state.

Relay Output		
Channels	8	
Output Type	Form A (SPST-NO)	
Contact Rating (Resistive Load)	5A 250V_{AC} (47~63Hz) 5A 30V_{DC}	
Operate Time	10ms max.	
Release Time	5ms max.	
Insulation Resistance	1,000 MΩ at 500V_{DC}	
Dielectric Strength	Between Open Contact	1000V_{AC} (1 min.)
	Between Coil and Contact	3000V_{AC} (1 min.)
Endurance	Mechanical	20,000,000 times min.
	Electrical	100,000 times min.
Power Consumption	1.235W max.	
Mechanical		
Dimensions (W×L×H)	33mm × 87mm × 107mm	

1.4.9 USB-2084

The USB-2084 is an 8-channel counters module with the USB interface. It provides a variety of measurement applications, such as measuring a number of time-related quantities, counting events or totalizing, and monitoring position with quadrature encoders. In addition, a digital filter is used to eliminate the effects of noise, and the filter's parameters are adjustable by software.

Counter		
Channels	4 channel counter type Up/Down 4 channel counter type Dir/Pulse 4 channel counter type A/B Phase 8 channels for counter type Up and Frequency	
Input Type	Up, Frequency, Up/Down, Dir/Pulse, A/B Phase	
Resolution	32 bit	
Input Frequency	500kHz maximum	
Digital Noise Filter	1~32767uS (Software programmable)	
Frequency Accuracy	±0.4%	
Isolated Input Level	On Voltage Level	$+4.5V_{DC}$~$+30V_{DC}$
	Off Voltage Level	$+1V_{DC}$ maximum
Non-isolated Input Level	On Voltage Level	$+2V_{DC}$~$+5V_{DC}$
	Off Voltage Level	$0V_{DC}$~$+0.8V_{DC}$
Intra-Module Isolation, Field-to-Logic	2500 V_{DC}	
Individual Channel Configuration	Yes	
Power Consumption	1.11W maximum	
Mechanical		
Dimensions (W×L×H)	33mm × 102mm × 107mm	

1.5 Product Check List

The package includes the following items:

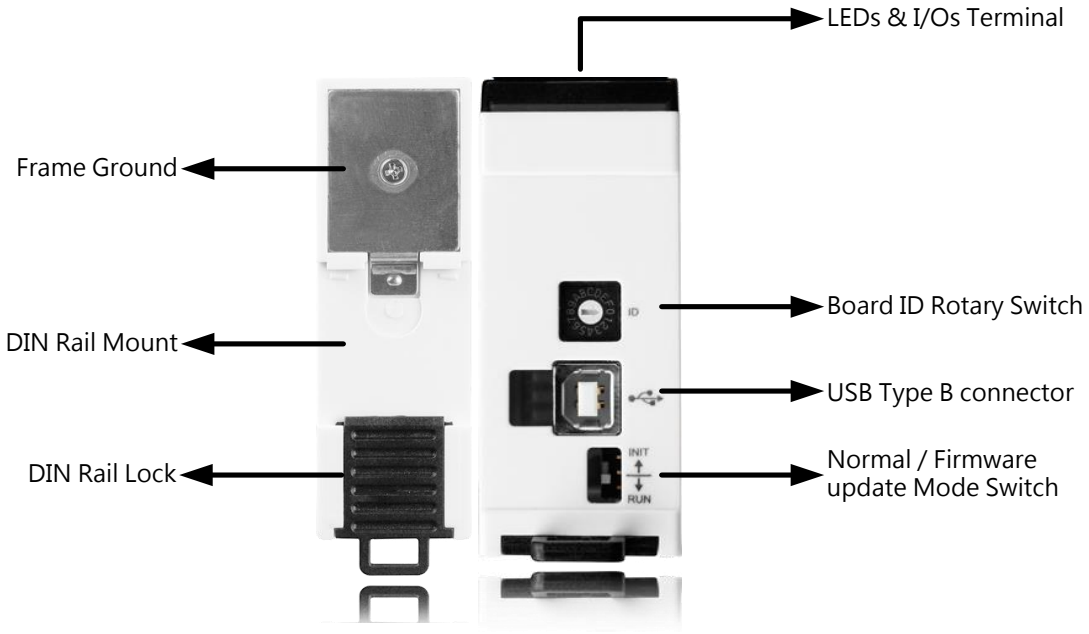
- One ICP DAS USB I/O module
- One Quick Start Guide
- One USB cable with lockable kit (CA-USB15)

It is highly recommended to read the Quick Start Guide first before using ICP DAS USB I/O modules. There is some useful information in the Quick Start Guide:

- How to install hardware and use utility

2 Hardware Information

2.1 Module Overview

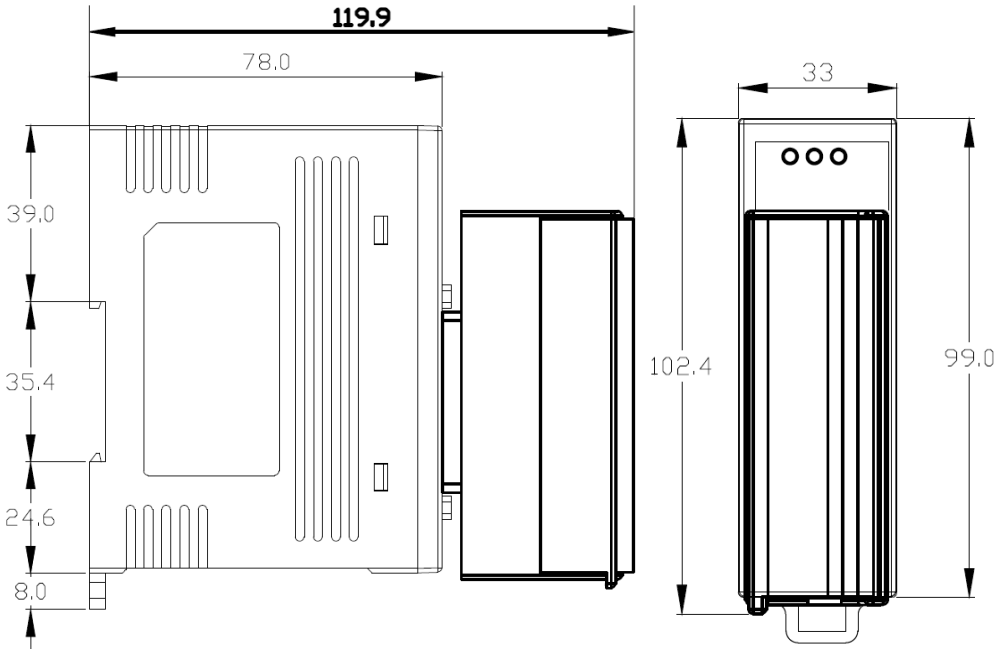


Board ID Rotary Switch	
0	: User defined (Software Programmable)
1 ~ 15	: Fix board ID

Normal / Firmware Update Mode Switch	
INIT	: Firmware update mode
RUN	: Normal mode

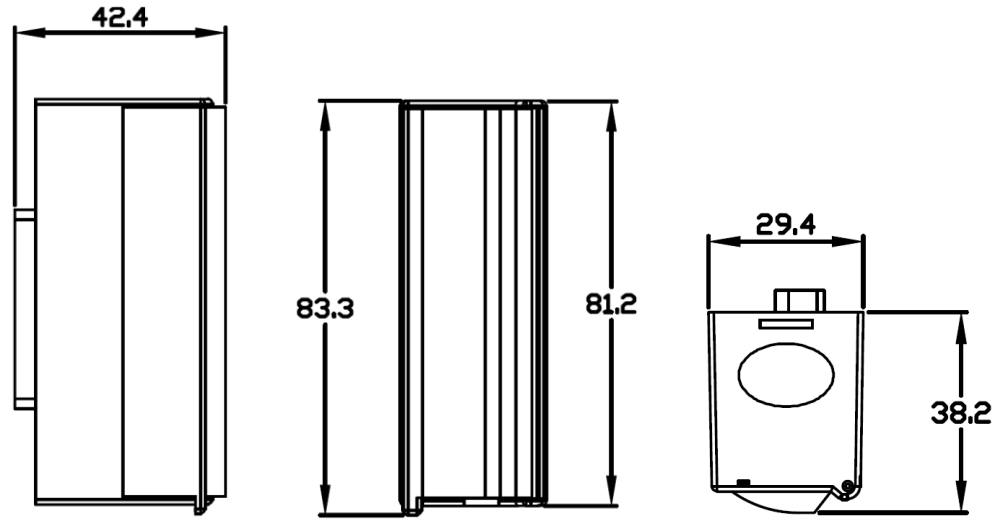
2.1.1 USB-2019

2.1.1.1 Body

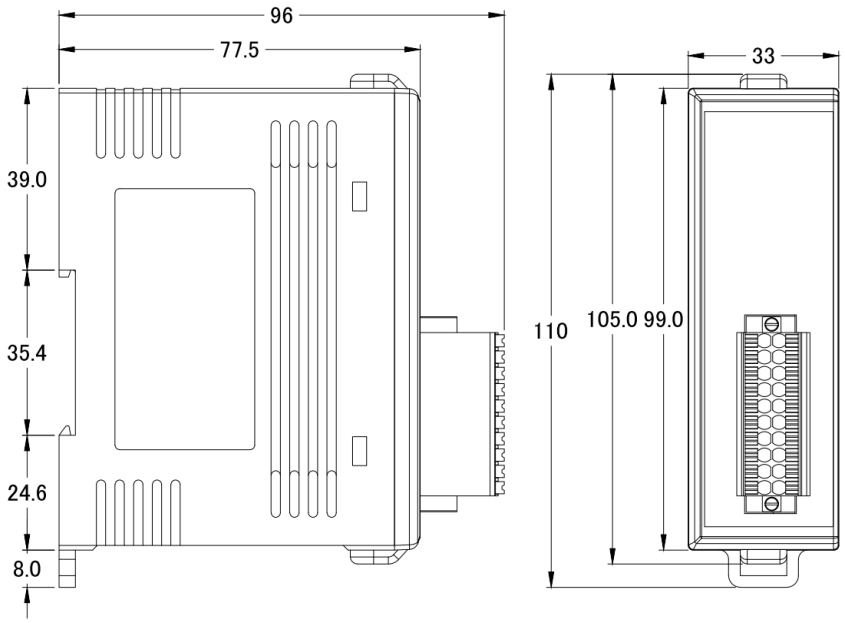


2.1.1.2 CN-1824

The CN-1824 is a connector transfers DB-25 connector to 18-pin terminal block to help user to wire. The dimension is shown as follow.



2.1.2 USB-2026 & USB-2045 & USB-2051 & USB-2055 &USB-2060



2.1.3 USB-2064

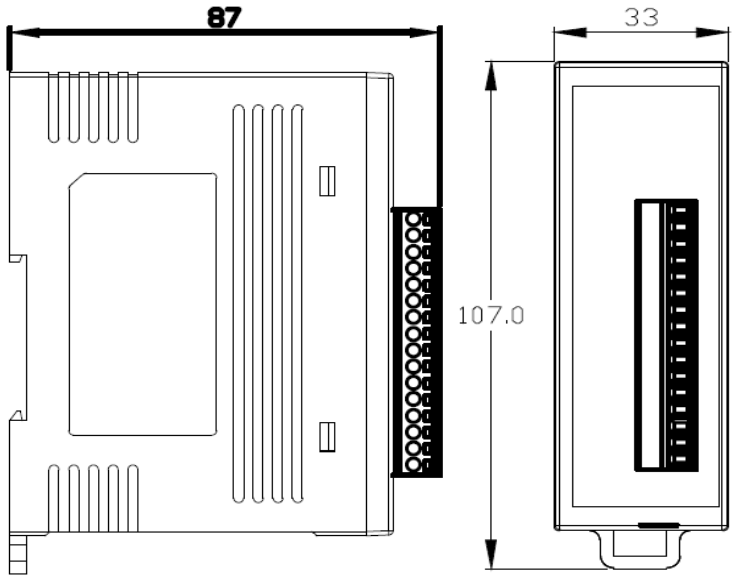


Figure 2-1, Figure 2-2 The USB-2084 left and front side view

2.1.4 USB-2084

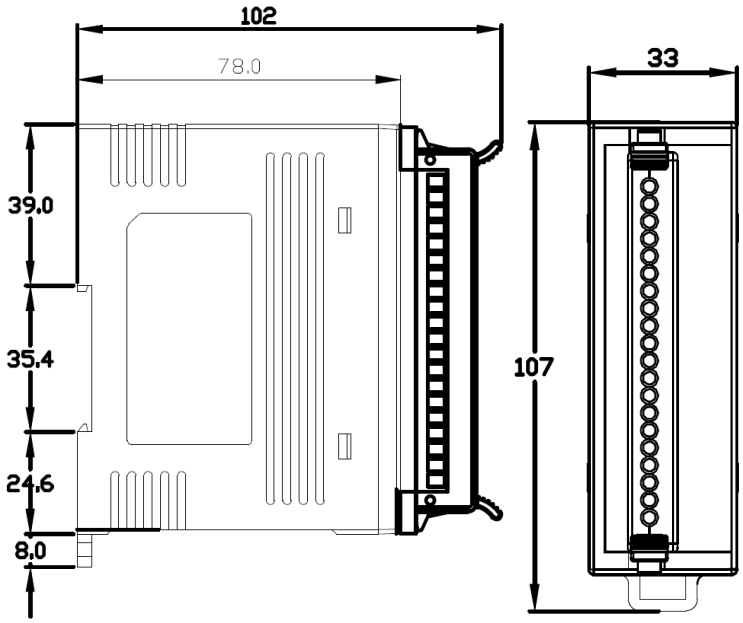


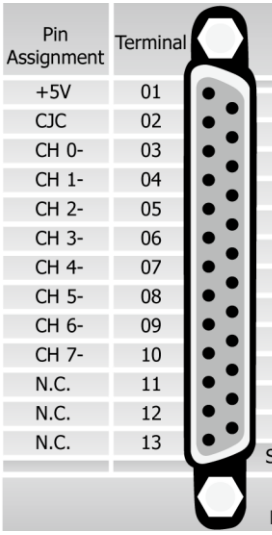
Figure 2-3, Figure 2-4 The USB-2084 left and front side view

2.1.5 CA-USB15

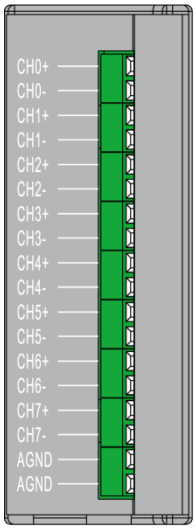
2.2 Connector Pin Assignment

2.2.1 USB-2019

Pin Assignment	Terminal No.		Terminal No.	Pin Assignment
+5V	01		14	AGND
CJC	02		15	CH 0+
CH 0-	03		16	CH 1+
CH 1-	04		17	CH 2+
CH 2-	05		18	CH 3+
CH 3-	06		19	CH 4+
CH 4-	07		20	CH 5+
CH 5-	08		21	CH 6+
CH 6-	09		22	CH 7+
CH 7-	10		23	N.C.
N.C.	11		24	N.C.
N.C.	12		25	N.C.
N.C.	13		Shield	F.G.



25-pin Female D-Sub Connector

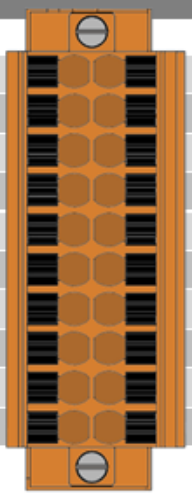


Pin Assignment Name
CH 0+
CH 0-
CH 1+
CH 1-
CH 2+
CH 2-
CH 3+
CH 3-
CH 4+
CH 4-
CH 5+
CH 5-
CH 6+
CH 6-
CH 7+
CH 7-
AGND
AGND

Signal	Description
AGND	Analog input ground
CH(N)+	Analog input channel N positive reference.
CH(N)-	Analog input channel N negative reference.

2.2.2 USB-2026

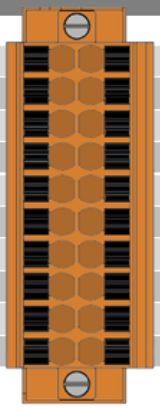
Pin Assignment	Terminal No.		Terminal No.	Pin Assignment
AIN0+	1		11	AIN0-
AIN1+	2		12	AIN1-
AIN2+	3		13	AIN2-
AIN3+	4		14	AIN3-
AIN4+	5		15	AIN4-
AOUT0+	6		16	AGND
AOUT1+	7		17	AGND
DI0	8		18	DO0
DI1	9		19	DO1
DGND	10		20	DGND



Signal	Description
AGND	Analog input ground
AI(N)+	Analog input channel N positive reference.
AI(N)-	Analog input channel N negative reference.
AO(N)+	Analog output channel N positive reference.
AO(N)-	Analog output channel N negative reference.
DGND	The Ground of Current Path of Digital input/output
DI(N)	The Digital Input for Channel N
DO(N)	The Digital Output for Channel N

2.2.3 USB-2045

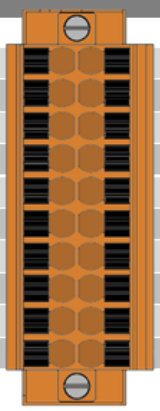
Pin Assignment	Terminal No.		Pin Assignment
Ext.GND	1	11	Ext.GND
DO0	2	12	DO8
DO1	3	13	DO9
DO2	4	14	DO10
DO3	5	15	DO11
DO4	6	16	DO12
DO5	7	17	DO13
DO6	8	18	DO14
DO7	9	19	DO15
Ext.PWR	10	20	Ext.PWR



Signal	Description
DO(N)	The Digital Output for Channel N
Ext.PWR	The Power Source Input Pin
Ext.GND	The Ground of Power Source Input Pin

2.2.4 USB-2051

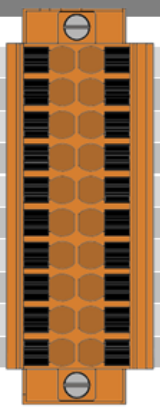
Pin Assignment	Terminal No.		Pin Assignment
Ext.GND	1	11	Ext.GND
DO0	2	12	DO8
DO1	3	13	DO9
DO2	4	14	DO10
DO3	5	15	DO11
DO4	6	16	DO12
DO5	7	17	DO13
DO6	8	18	DO14
DO7	9	19	DO15
Ext.PWR	10	20	Ext.PWR



Signal	Description
DI(N)	The Digital Input for Channel N
DI.GND	The Ground of Current Path of Dry Contact
DI.COM	The Common Pin for Wet Contact

2.2.5 USB-2055

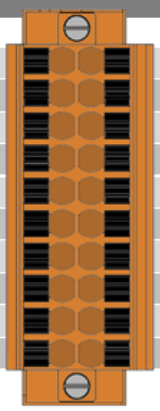
Pin Assignment	Terminal No.		Pin Assignment
Ext.GND	1	11	DI.GND
DO0	2	12	DI0
DO1	3	13	DI1
DO2	4	14	DI2
DO3	5	15	DI3
DO4	6	16	DI4
DO5	7	17	DI5
DO6	8	18	DI6
DO7	9	19	DI7
Ext.PWR	10	20	DI.COM



Signal	Description
DI(N)	The Digital Input for Channel N
DI.GND	The Ground of Current Path of Dry Contact
DI.COM	The Common Pin for Wet Contact
DO(N)	The Digital Output for Channel N
Ext.PWR	The Power Source Input Pin
Ext.GND	The Ground of Power Source Input Pin

2.2.6 USB-2060

Pin Assignment	Terminal No.		Pin Assignment
RL0 NO	1	11	DI.GND
RL0 COM	2	12	DI0
RL1 NO	3	13	DI1
RL1 COM	4	14	DI2
RL2 NO	5	15	DI3
RL2 COM	6	16	DI4
RL3 NO	7	17	DI5
RL3 COM	8	18	DI.COM
RL4 NO	9	19	RL5 NO
RL4 COM	10	20	RL5 COM



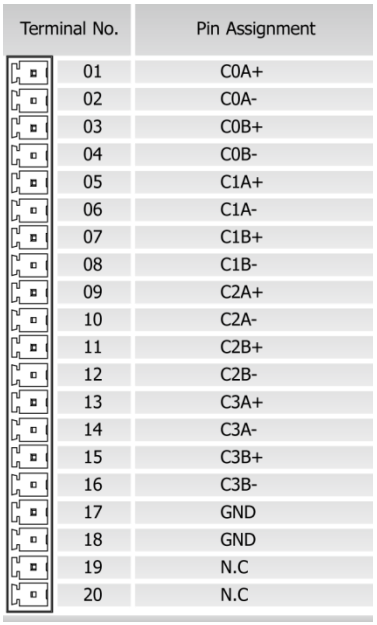
Signal	Description
DI(N)	The Digital Input for Channel N
DI.GND	The Ground of Current Path of Dry Contact
DI.COM	The Common Pin for Wet Contact
RL(N) NO	The NO pin of relay(N)
RL(N) COM	The COM pin of relay(N)

2.2.7 USB-2064

Terminal No.	Pin Assignment
01	RL0 NO
02	RL0 COM
03	RL1 NO
04	RL1 COM
05	RL2 NO
06	RL2 COM
07	RL3 NO
08	RL3 COM
09	RL4 NO
10	RL4 COM
11	RL5 NO
12	RL5 COM
13	RL6 NO
14	RL6 COM
15	RL7 NO
16	RL7 COM

Signal	Description
RL(N) NO	The NO pin of relay(N)
RL(N) COM	The COM pin of relay(N)

2.2.8 USB-2084

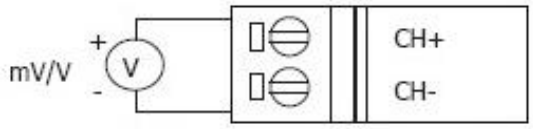
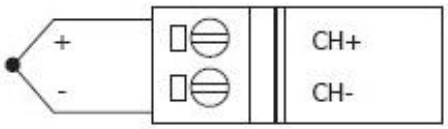
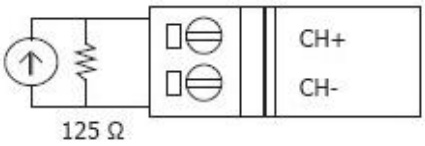


Terminal No.	Pin Assignment
01	C0A+
02	C0A-
03	C0B+
04	C0B-
05	C1A+
06	C1A-
07	C1B+
08	C1B-
09	C2A+
10	C2A-
11	C2B+
12	C2B-
13	C3A+
14	C3A-
15	C3B+
16	C3B-
17	GND
18	GND
19	N.C
20	N.C

Signal	Description
GND	Ground pin for non-isolated connection
C(N)A+	Counter positive signal channel for pair A
C(N)A-	Counter negative signal channel for pair A
C(N)B+	Counter positive signal channel for pair B
C(N)B-	Counter negative signal channel for pair B
N.C	No connection on this pin

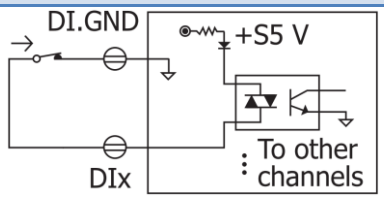
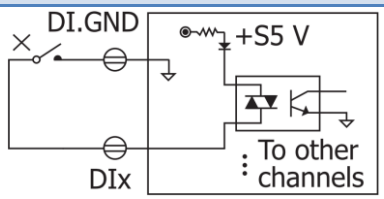
2.3 Wiring

2.3.1 USB-2019

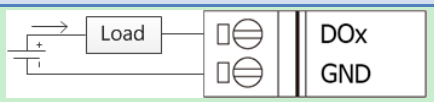
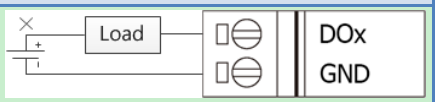
Voltage Input	Thermocouple Input
	
Current Input	
	
Note: When connecting to current source, an external 125Ω resistor is required.	

2.3.2 USB-2026

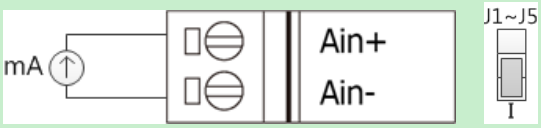
2.3.2.1 Digital Input

Input	ON	OFF
Dry Contact		

2.3.2.2 Digital Output

Output	ON	OFF
Load		

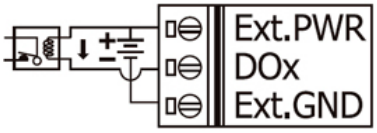
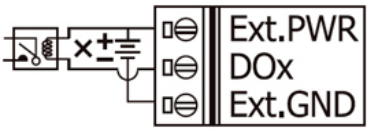
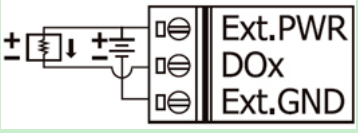
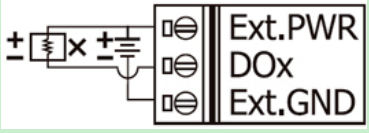
2.3.2.3 Analog Input

Voltage Input
Current Input


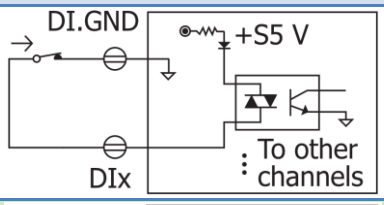
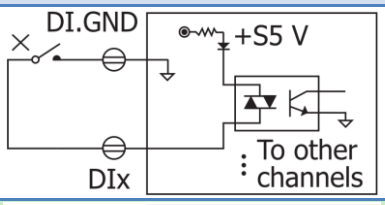
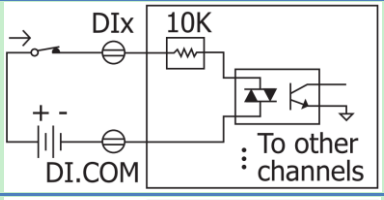
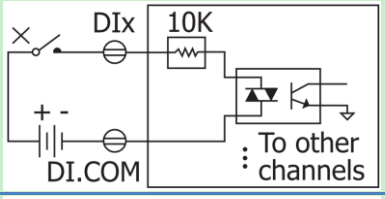
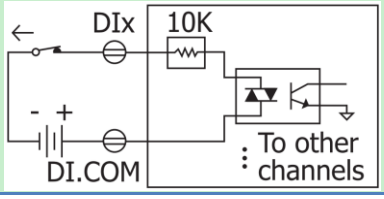
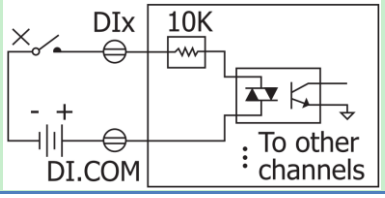
2.3.2.4 Analog Output

Voltage Input
Current Input

2.3.3 USB-2045

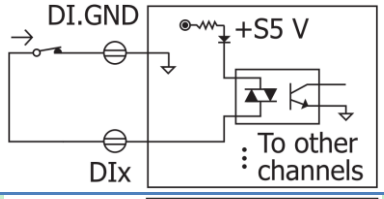
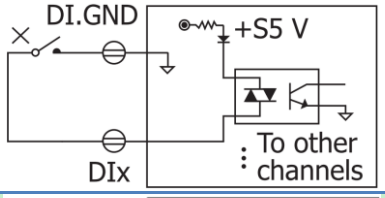
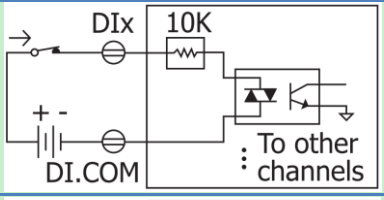
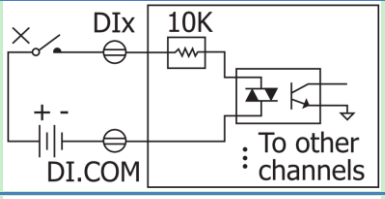
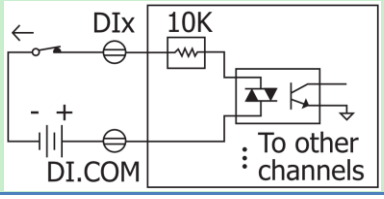
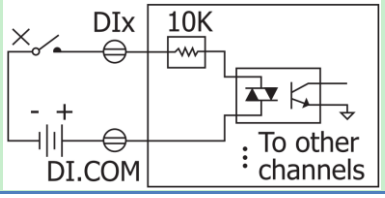
Output	ON	OFF
Drive Relay		
Resistance Load		

2.3.4 USB-2051

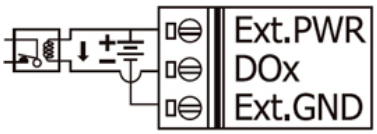
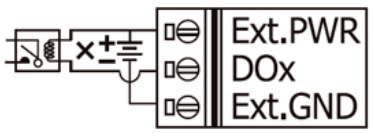
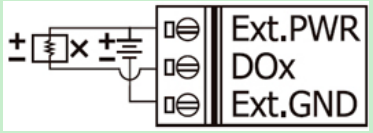
Input	ON	OFF
Dry Contact		
Wet Contact (Sink)		
Wet Contact (Source)		

2.3.5 USB-2055

Digital Input

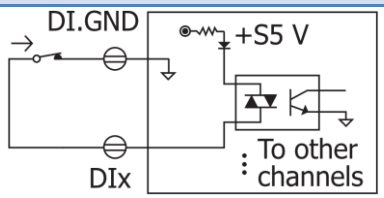
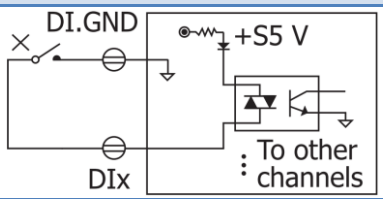
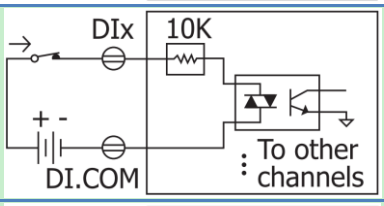
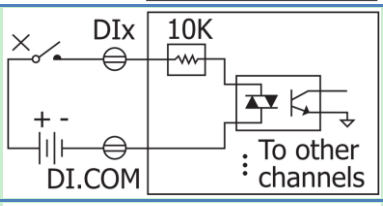
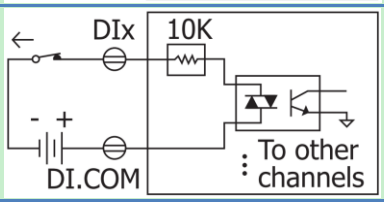
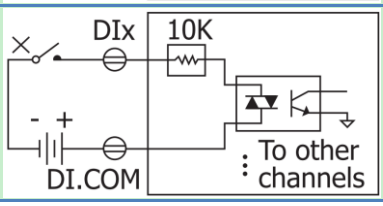
Input	ON	OFF
Dry Contact		
Wet Contact (Sink)		
Wet Contact (Source)		

Digital Output

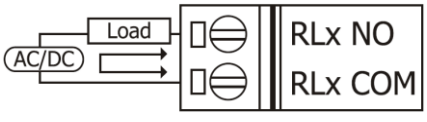
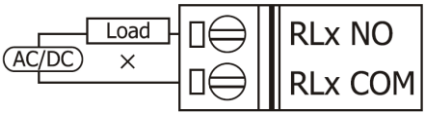
Output	ON	OFF
Drive Relay		
Resistance Load		

2.3.6 USB-2060

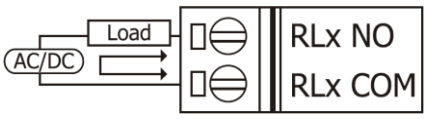
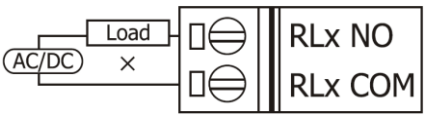
Digital Input

Input	ON	OFF
Dry Contact		
Wet Contact (Sink)		
Wet Contact (Source)		

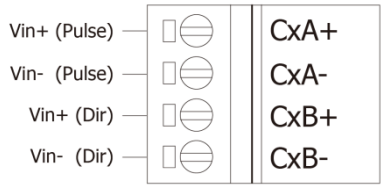
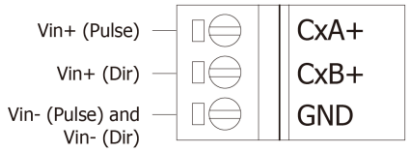
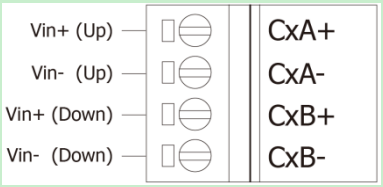
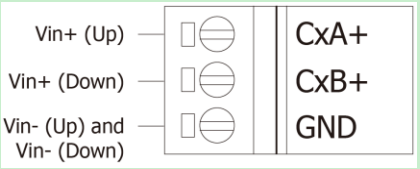
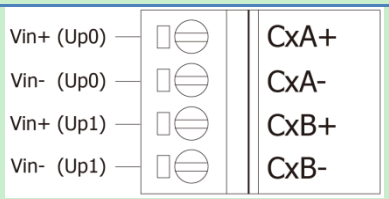
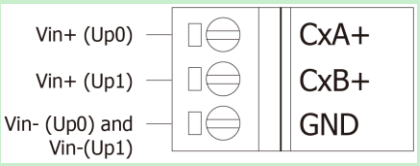
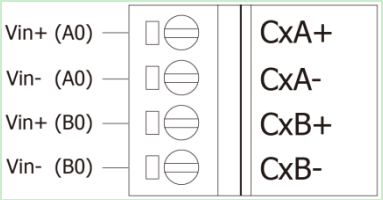
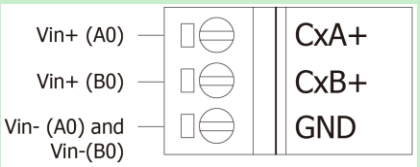
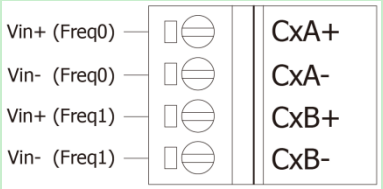
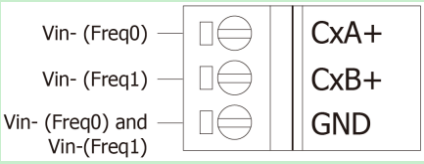
Relay Output

Output Type	ON State LED ON Readback as 1	OFF State LED OFF Readback as 0
Relay Contact		

2.3.7 USB-2064

Output Type	ON State LED ON Readback as 1	OFF State LED OFF Readback as 0
Relay Contact		

2.3.8 USB-2084

Input Mode	Isolated	Non-isolated
Dir/Pulse		
Up/Down		
Up		
A/B Phase (Quadrant)		
Frequency		

2.4 Hardware Configuration

The ICP DAS USB series I/O modules provide two basic configurations of hardware to configure board ID and enable firmware update functionality.

2.4.1 Board ID

The board ID is used to identify two modules with same product number connected to computer. When two more modules with same product number are connected, each of them must be set to different board ID to prevent conflict and unexpected errors. The board ID can be configured by the rotary switch. The location of the rotary switch is shown in figure 2-15. The value of board ID can be configured from 1 ~ 15 by hardware, and can be configured from 16 ~ 127 by software when switched to 0.

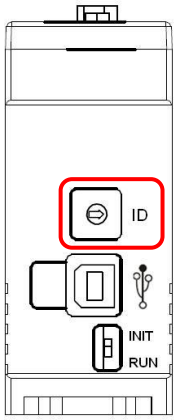


Figure 2-5 The hardware setting for board ID

2.4.2 Firmware Update

The ICP DAS USB series I/O modules provide firmware updateable functionality. Users can update firmware if latest firmware released. The switch setting is shown in figure 2-16. The INIT side of the switch means firmware update mode, run side means normal operation.

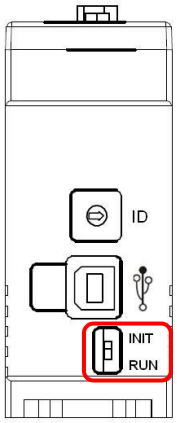
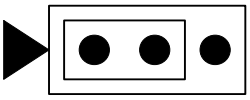
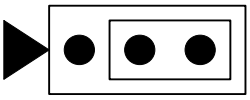


Figure 2-6 The hardware setting for enabling firmware update functionality

2.4.3 USB-2019

2.4.3.1 Hardware Watchdog

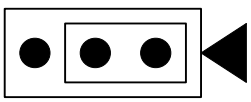
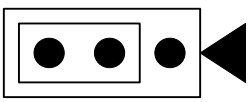
The USB-2019 has a build-in hardware watchdog. It is recommended to enable this functionality. The hardware watchdog can be set by jumper JP1. The watchdog setting is enabled by default.

Jumper	Setting			
JP1	Enable (Default)		Disable	

2.4.4 USB-2026

2.4.4.1 Hardware Watchdog

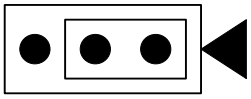
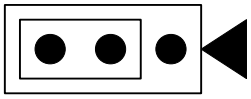
The USB-2026 has a build-in hardware watchdog. It is recommended to enable this functionality. The hardware watchdog can be set by jumper JP1. The watchdog setting is enabled by default.

Jumper	Setting			
JP1	Enable (Default)		Disable	

2.4.5 USB-2045

2.4.5.1 Hardware Watchdog

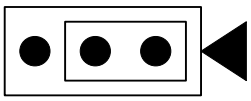
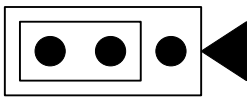
The USB-2045 has a build-in hardware watchdog. It is recommended to enable this functionality. The hardware watchdog can be set by jumper JP1. The watchdog setting is enabled by default.

Jumper	Setting			
JP1	Enable (Default)		Disable	

2.4.6 USB-2051

2.4.6.1 Hardware Watchdog

The USB-2051 has a build-in hardware watchdog. It is recommended to enable this functionality. The hardware watchdog can be set by jumper JP1. The watchdog setting is enabled by default.

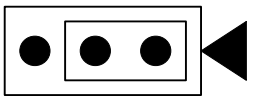
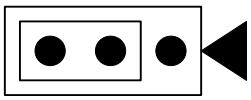
Jumper	Setting			
JP1	Enable (Default)		Disable	

2.4.7 USB-2055

2.4.7.1 Hardware Watchdog

The USB-2055 has a build-in hardware watchdog. It is recommended to enable this functionality. The hardware watchdog can be set by jumper JP1. The watchdog setting is enabled by default.

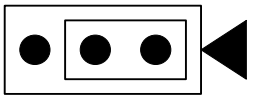
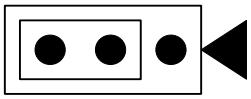
Jumper	Setting

JP1	Enable (Default)		Disable	

2.4.8 USB-2060

2.4.8.1 Hardware Watchdog

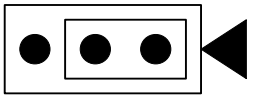
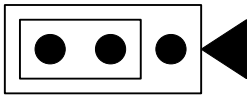
The USB-2060 has a build-in hardware watchdog. It is recommended to enable this functionality. The hardware watchdog can be set by jumper JP1. The watchdog setting is enabled by default.

Jumper	Setting			
JP1	Enable (Default)		Disable	

2.4.9 USB-2064

2.4.9.1 Hardware Watchdog

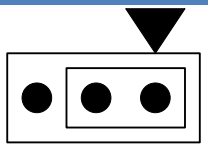
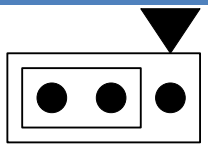
The USB-2064 has a build-in hardware watchdog. It is recommended to enable this functionality. The hardware watchdog can be set by jumper JP1. The watchdog setting is enabled by default.

Jumper	Setting			
JP1	Enable (Default)		Disable	

2.4.10 USB-2084

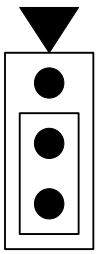
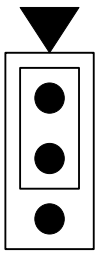
2.4.10.1 Hardware Watchdog

The USB-2084 has build-in hardware watchdog. It is recommended to enable this functionality. The hardware watchdog can be set by jumper JP1. The watchdog setting is enabled by default.

Jumper	Setting			
JP1	Enable (Default)		Disable	

2.4.10.2 Isolated/Non-isolated (TTL)

The USB-2084 has two kind of inputs, isolated and non-isolated (TTL), for different input signals. Users can switch jumper setting on the USB-2084 board for appropriate signal. These jumpers are located within JP4~JP11. The jumper settings are listed in the following table. The isolated input is set by default.

Jumper	Counter	Jumper setting			
JP4	A0	Isolated input (Default)		Non-isolated input (TTL input)	
JP5	B0				
JP6	A1				
JP7	B1				
JP8	A2				
JP9	B2				
JP10	A3				
JP11	B3				

2.5 LED Indicators

The ICP DAS USB series I/O modules have two modes, normal and firmware update, are described in previous section. Each mode has own LED way of indication. The LED indications for two modes are shown below.

2.5.1 Normal Operation

LED Indicators	LED Status	Causes
PWR (Red)	Blink	HW WDT triggered
	Solid	Normal Operation
	Off	Power Off
RUN (Green)	Blink	USB Bus Communicating
	Off	USB Bus Idle
ERR (Yellow)	Blink (Less frequent)	Warning (**Does not** affect the operation)
	Blink	Minor Error (**Does not** affect the operation)
	Solid	Major Error (**Does** affect the operation)
	Off	No Error

2.5.2 Firmware update

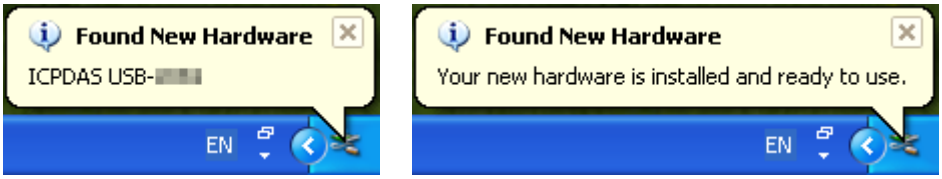
LED Indicators	LED Status	Causes
ALL	Blink	Waiting for Firmware to update

3 Installation

3.1 Hardware

3.1.1 Connecting to ICP DAS USB series I/O module

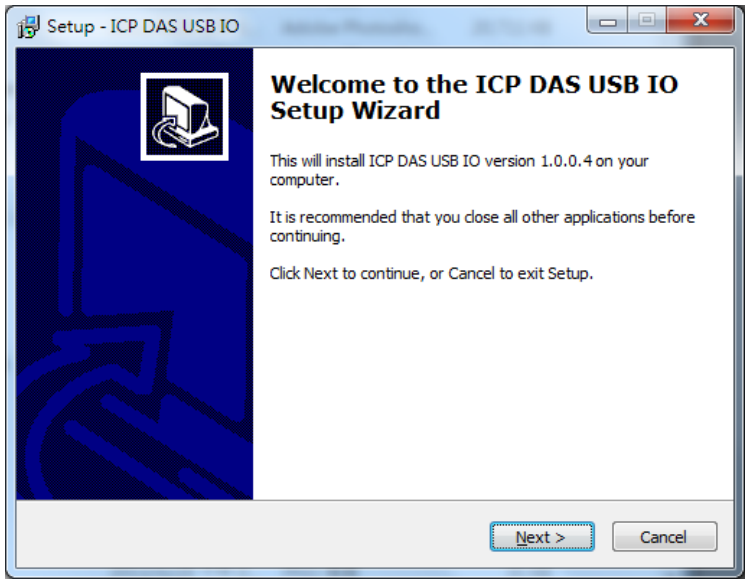
1. Turning on the PC you are preparing to configure ICP DAS USB I/O modules.
2. Connecting the ICP DAS USB series I/O modules to USB port on the PC.
3. Once you first time connect the USB I/O module to PC. There will be few messages in system bar in bottom right side to inform new hardware is detected and installed successfully. After the message is shown as figure 3-2, then the ICP DAS USB series I/O modules are ready to use.



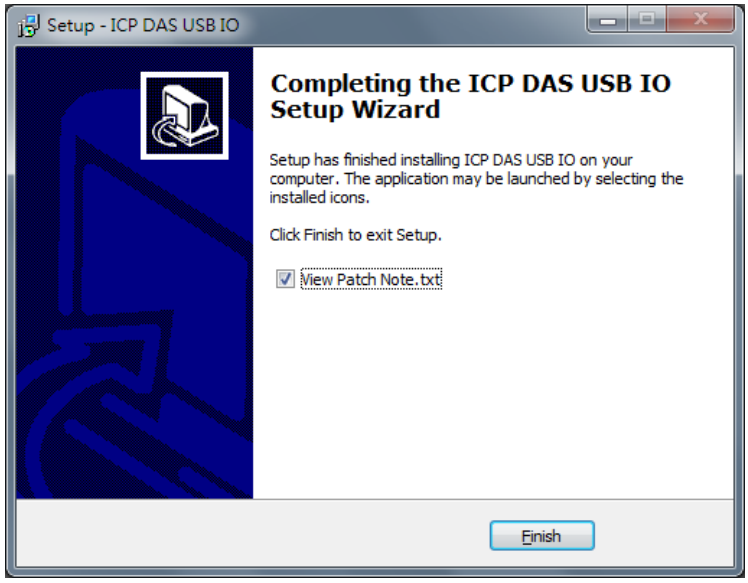
***NOTE: It is strongly recommended that users use the cable we provided to connect to USB I/O.**

3.2 Software

The software installer includes libraries, samples and Utility, and can be found in web site. You can install the package by double clicking the file "ICPDAS USB IO X.X.X.exe". Then follow the instruction during installation.


Setup - ICP DAS USB IO
Welcome to the ICP DAS USB IO Setup Wizard
This will install ICP DAS USB IO version 1.0.0.4 on your computer.
It is recommended that you close all other applications before continuing.
Click Next to continue, or Cancel to exit Setup.
Next >
Cancel

After the installation, the window will indicate the installation has completed as the figure below. Users can check or ignore the patch note in this step.

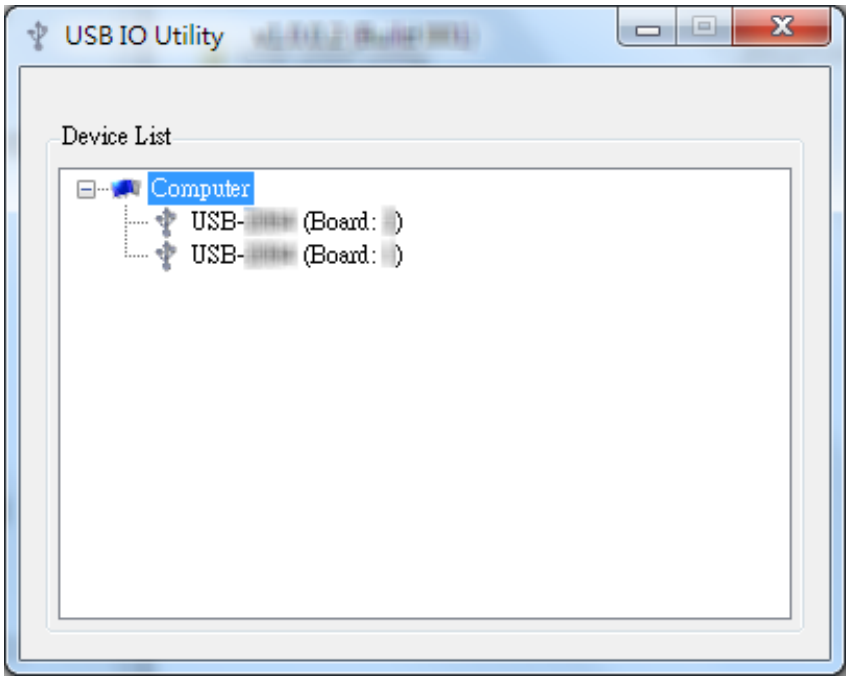

Setup - ICP DAS USB IO
Completing the ICP DAS USB IO Setup Wizard
Setup has finished installing ICP DAS USB IO on your computer. The application may be launched by selecting the installed icons.
Click Finish to exit Setup.
View Patch Note.txt
Finish

3.2.1 Utility

The USB IO Utility provides a simple way to test and acquire data easily and instantly for all ICP DAS USB series I/O modules without programming. You will find this program in “Start\Programs\ICPDAS\USB IO\USB IO Utility” or the path “C:\ICPDAS\USB IO\USB IO Utility\USB IO Utility.exe”.

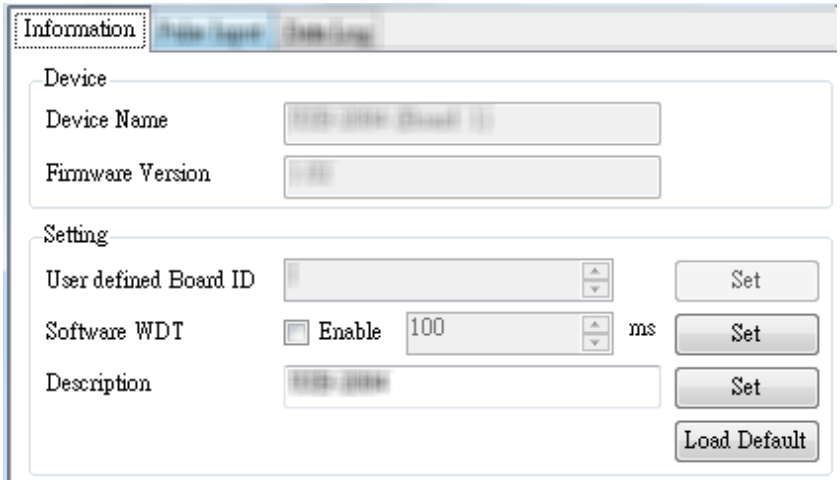
When users open USB IO Utility, the all ICP DAS USB series I/O modules connected to PC are listed in “Device List” as figure 3-5. The utility will scan the ICP DAS USB series

I/O modules automatically. The module in the “Device List” will be removed when pull-off from PC and added when plug-in to PC.



To access the ICP DAS USB I/O module user can double click the module listed in “Device List”, and then you will see another form come out. There are several function pages, information and I/O pages, in the device form. In the information page of the device form, it is used to configure basic system parameters. And in the I/O page, it is used to access I/O data and configure parameters. There will be a data log page if module supports this functionality.

3.2.1.1 Information Page



- **Device Name**

  The name of the opened device.

- **Firmware Version**

  The firmware version of the opened device.

- **User Defined Board ID**

  The board ID of the opened device. The value can configure when switched the board ID to "0" by the rotary switch.

  Note: The valid range of this ID is from 16 to 127.

- **Software WDT**

  The software watch dog timer of the opened device. The value enables the functionality to monitor module still alive or not. When enable the watch dog timer, computer and module will send SYNC packet each other. When communication is failure, software WDT also provides functionality to output safety value if the module has output capability.

  Note: The valid range of this value is 100 ms ~ 30 minute.

- **Description**

  The description of the opened device. This item helps user to identify module.

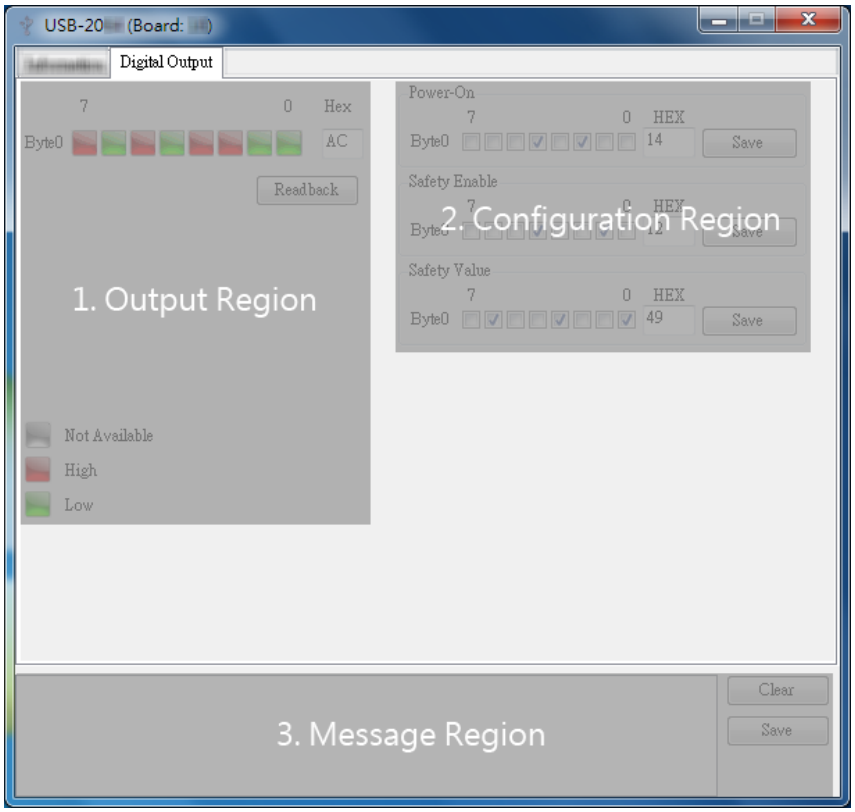
  Note: The maximum characters of the description are 32.

- **Load Default**

  This function is used to restore module to default setting (factory setting).

3.2.1.2 Digital Output

In the I/O page of the digital output, the digital output value and module configuration can be read or written in this page. The detail of all items in this form will be introduced in this section.



- **Output Region**

  The output region is used to wirte the output value to the module. The output value can be writen by the LED to each channel or the hex to all channels. Users can also readback the output value in this region.

  NOTE: The readback value is the current output value in the module if the module does not support diagnostic functionality. If the module does support diagnostic, the value for readback will be the actual output value in this module.

- **Configuration Region**

  All I/O related configurations can be set in this region.

  - **Power-On**

    The power-on configuration.

- **Safety Enable**

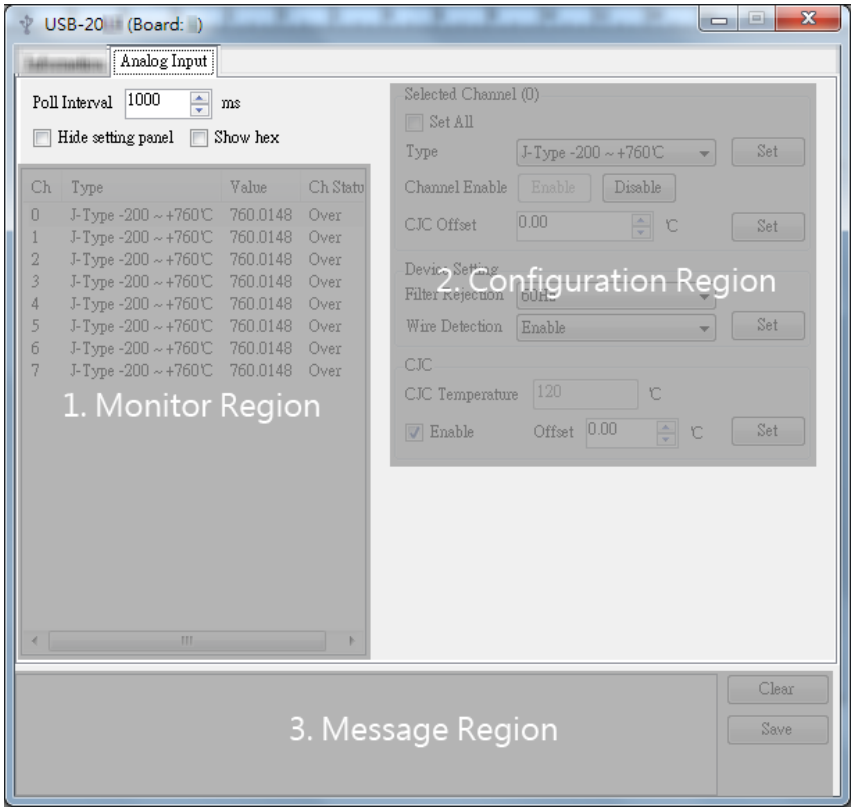
  The safety enable configuration.

- **Safety Value**

  The safety value.

3.2.1.3 Analog Input

In the I/O page of the AI, the real-time value and module configuration can be read or written in this page. The detail of all items in this form will be introduced in this section.


USB-20 (Board:)
Analog Input
Poll Interval 1000 ms
Hide setting panel
Show hex
Ch Type Value Ch Statu
0 J-Type -200 ~ +760℃ 760.0148 Over
1 J-Type -200 ~ +760℃ 760.0148 Over
2 J-Type -200 ~ +760℃ 760.0148 Over
3 J-Type -200 ~ +760℃ 760.0148 Over
4 J-Type -200 ~ +760℃ 760.0148 Over
5 J-Type -200 ~ +760℃ 760.0148 Over
6 J-Type -200 ~ +760℃ 760.0148 Over
7 J-Type -200 ~ +760℃ 760.0148 Over
1. Monitor Region
Selected Channel (0)
Set All
Type J-Type -200 ~ +760℃ Set
Channel Enable Enable Disable
CJC Offset 0.00 ℃ Set
2. Configuration Region
Wire Detection Enable Set
CJC
CJC Temperature 120 ℃
Enable Offset 0.00 ℃ Set
3. Message Region
Clear
Save

- **Polling Interval**

  This value is the period to poll data to the USB I/O module.

  Note: The valid value is 100 ~ 5000ms.

- **Hide Setting Panel**

  Hiding the I/O configuration panel.

- **Show Hex**

  Converting the I/O value from decimal to hexadecimal.

- **Monitor Region**

  The I/O related data and configurations will be listed here. Users can select the channel to configure in the "I/O Monitor Region". The setting of this selected channel will show in "I/O configuration region".

- **Configuration Region**

  All I/O related configurations can be set in this region. This region is divided into two parts, channel and module related setting. The channel related setting is in the "Selected Channel". The rest are module related settings.

  - **Set All**

    All channels related setting will follow current selection.

  - **Type**

    The ICP DAS USB series I/O modules provide programmable input type for analog input. Users can set different type for each analog input channel. For more detail for type of analog input modules, please refer to Appendix A.1.

  - **Channel Enable**

    Enable / Disable channel.

  - **Channel CJC Offset**

    Setting the CJC offset for the specific channel. The behavior of the setting is the same as the CJC Offset, but it only affects specific channel.

    Note: The CJC offset can be any in the range of -40.96 to +40.95 °C.

  - **Filter Rejection**

    In order to remove the noise from the power supply, some analog input modules feature build-in noise filter. Two filters, 50Hz and 60Hz, are provided to remove noise generated from power source.

  - **Wire Detection**

    Enable / Disable the open-wire detection for thermocouple and 4~20 mA.

  - **CJC Enable**

    Enable / Disable the CJC (Cold-Junction Compensation).

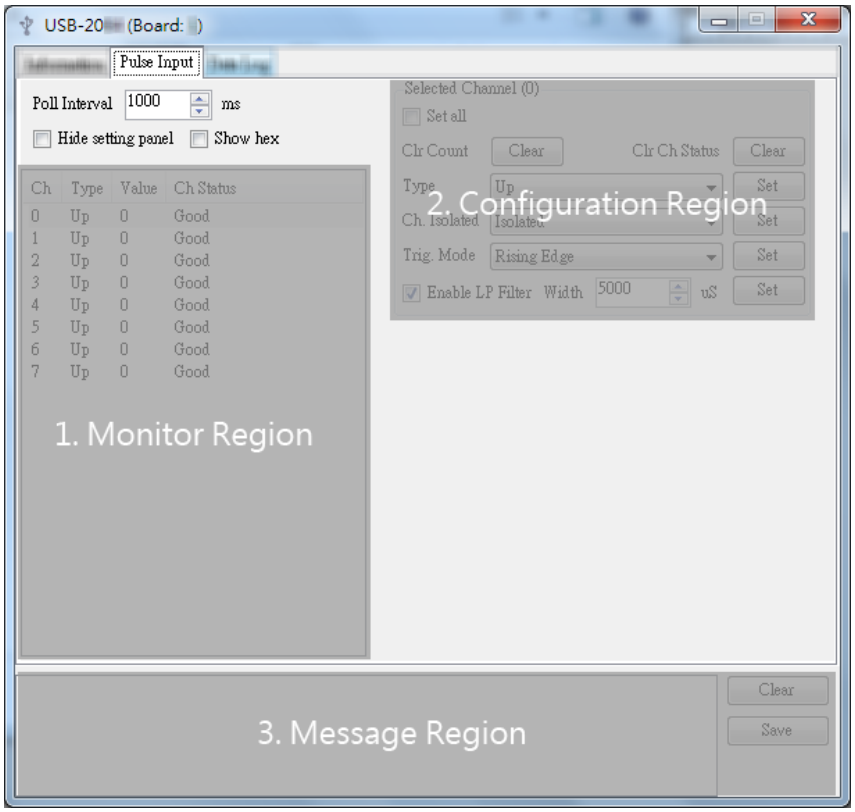
  - **CJC Offset**

    Setting the CJC offset value for all AI channels. The offset value is used to add or subtract the reading value. Changing of this value will not affect calibration, but will affect the reading value of temperature type.

    Note: The CJC offset can be any in the range of -40.96 to +40.95 °C.

3.2.1.4 Pulse Input

In the I/O page of the PI, the real-time value and module configuration can be read or written in this page. The detail of all items in this form will be introduced in this section.



- **Polling Interval**

  This value is the period to poll data to the USB I/O module.

  Note: The valid value is 100 ~ 5000ms.

- **Hide Setting Panel**

  Hiding the I/O configuration panel.

- **Show Hex**

  Converting the I/O value from decimal to hexadecimal.

- **Monitor Region**

  The I/O related data and configurations will be listed here. Users can select the channel to configure in the "I/O Monitor Region". The setting of this selected channel will show in "I/O configuration region".

- **Configuration Region**

All I/O related configurations can be set in this region. This region is divided into two parts, channel and module related setting. The channel related setting is in the "Selected Channel". The rest are module related settings.

- **Set All**

  All channels related setting will follow current selection.

- **Type**

  The ICP DAS USB series I/O modules provide programmable input type for pulse input. Users can set different type for each pulse input channel. For more detail for type of pulse input modules, please refer to Appendix A.3.

- **Clr Count**

  Clear counter value for specified channel.

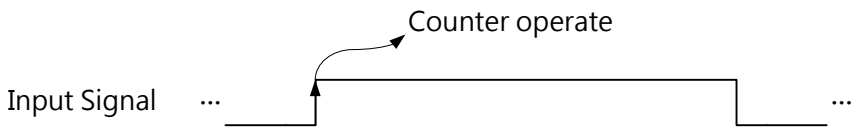
- **Ch. Isolated**

  The USB-2084 has isolated and non-isolated (TTL) inputs. To switch different input, two parts have to set as well. One is jumper JP4~JP10 described in 2.5.4.2, and the other is Ch. Isolated in Utility.

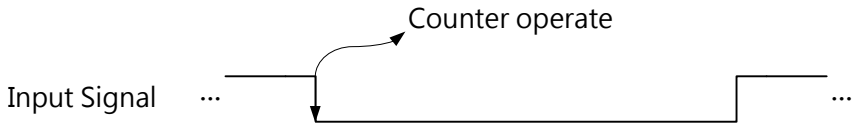
- **Trig. Mode**

  The USB-2084 has rising and falling edge trigger modes. The difference between rising and falling is the timing of counter operation. In rising edge trigger mode, counter will operate when the input signal from low to high level. In contrast, counter will operate when the input signal from high to low level in the falling edge trigger mode.

  Rising edge：



  Falling edge：



- **Enable LP Filter**

  To enable build-in digital low pass filter. The detail of this setting will be introduced in "LP Width" section.

- **LP Width**

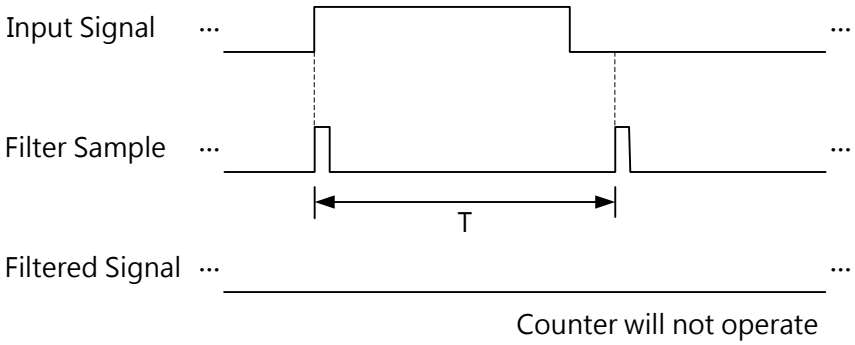
  The USB-2084 has three independent digital noise filters, LP0, LP1 and LP2, to remove noise. 8 counters share these three filters. The following table shows the relationship between filters and counters.

Channel	Low Pass Filter
A0	LP0
B0	LP0
A1	LP1
B1	LP1
A2	LP2
B2	LP2
A3	LP2
B3	LP2

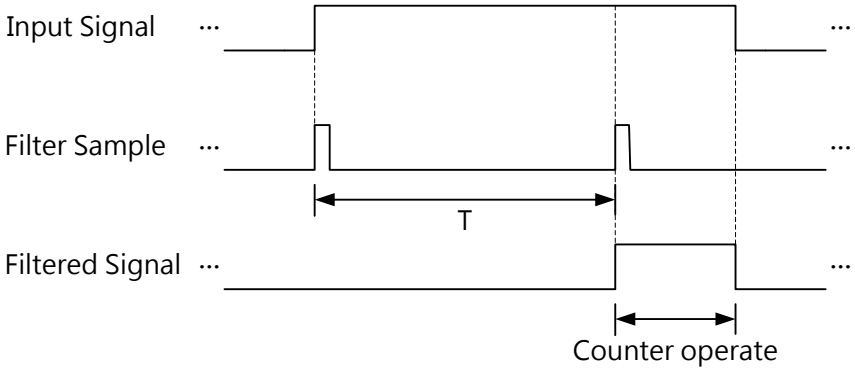
  The low pass filter width can be either disable or enable, and the width can be programmed from **1 to 32767 us**.

  The basic operation of filter is shown in following figure. The counter will operate when input signal hold on the same level during filter width.

(a) If the high width of the input signal is shorter then T, the counter will not operate. The input signal will be filtered. The time chart is shown as follow.



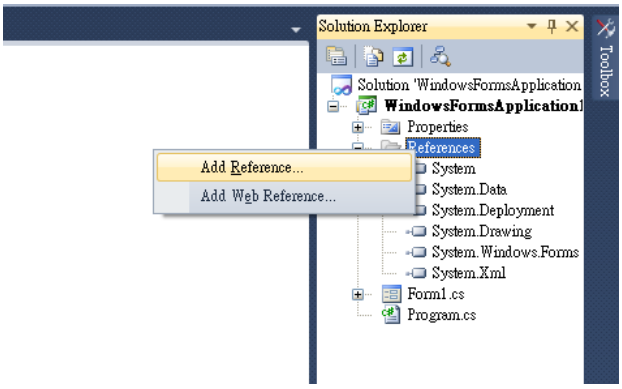
(b) If the high width of the input signal is grater then T, the counter will operate. The time chart is shown as follow.

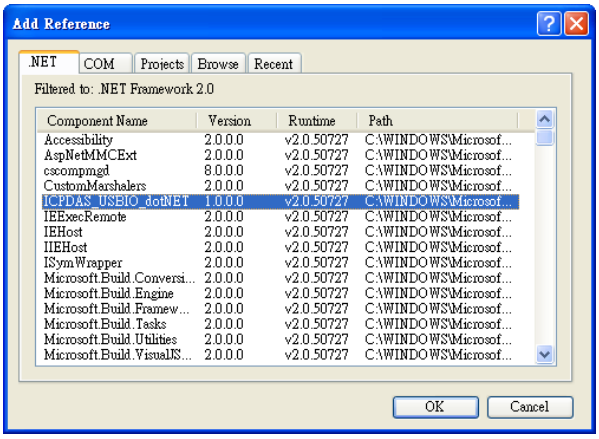


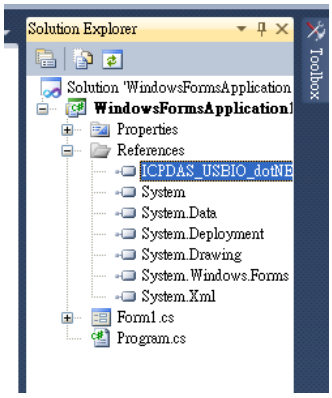
3.2.2 ICP DAS USB I/O Software Integration

The USB I/O libraries are the way to access ICP DAS USB series I/O modules. It supports various IDE like C#/VB.NET/VB/VC/BCB. Users can choose any IDE you familiar with. Before starting up project, you need to do some configuration to integrate the SDK into your IDE. The following section will indicate you how to integrate the SDK into your IDE.

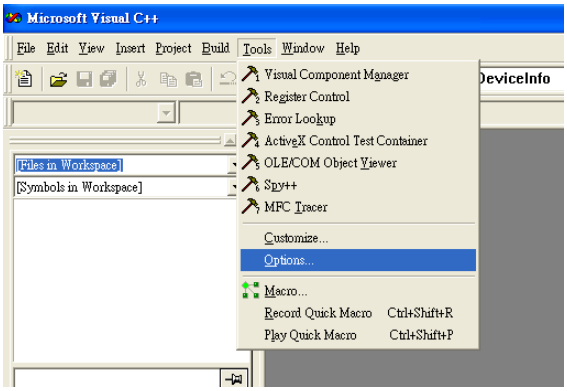
3.2.2.1 .NET

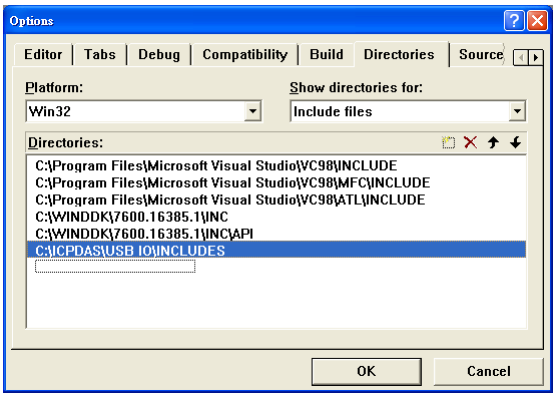


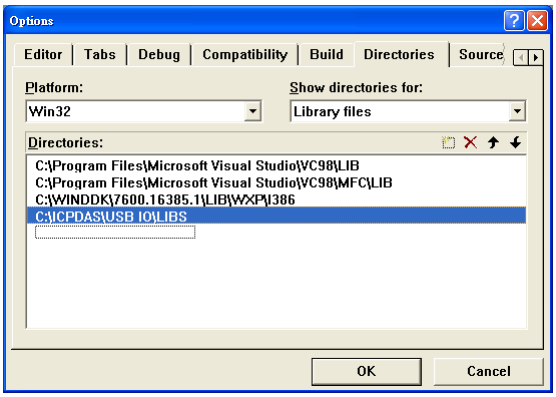




3.2.2.2 VC



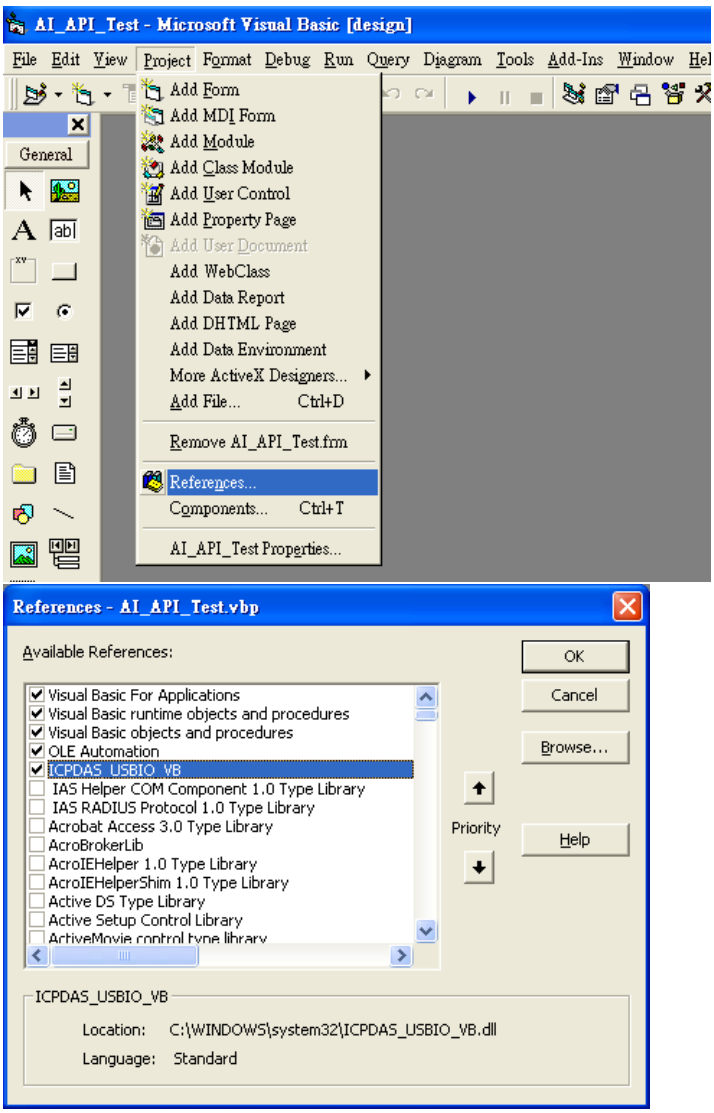




3.2.2.3 BCB

This section is left blank intentionally.

3.2.2.4 VB



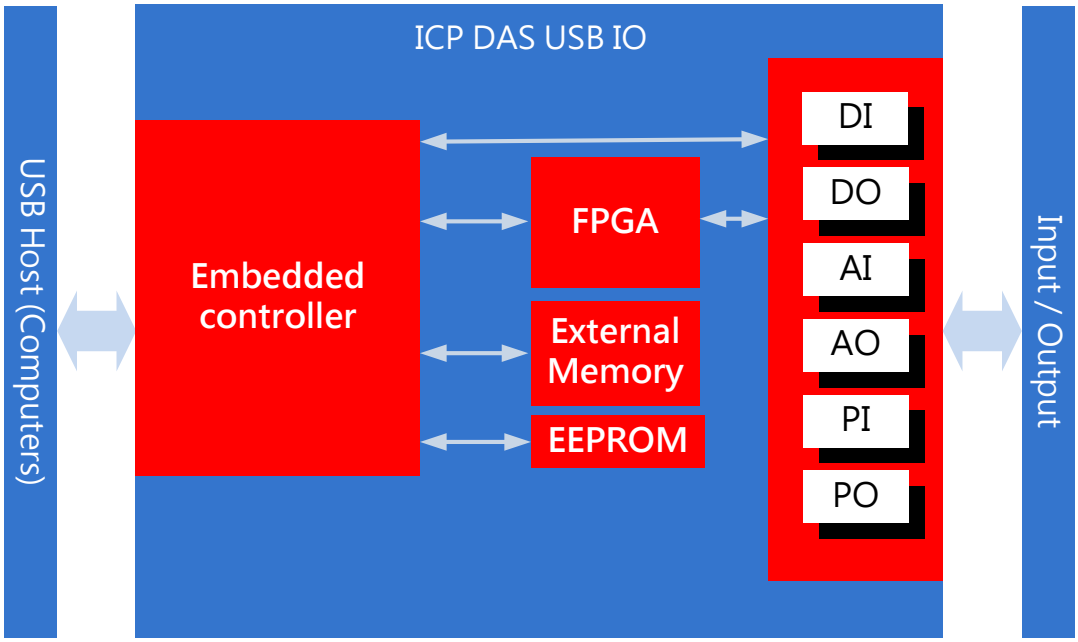
3.2.3 Samples

There are several samples to help user to develop project smoothly. The samples can be found in “Start\Programs\ICPDAS\USB IO\Samples” or the path “C:\ICPDAS\USB IO\Samples“.

4 Operation

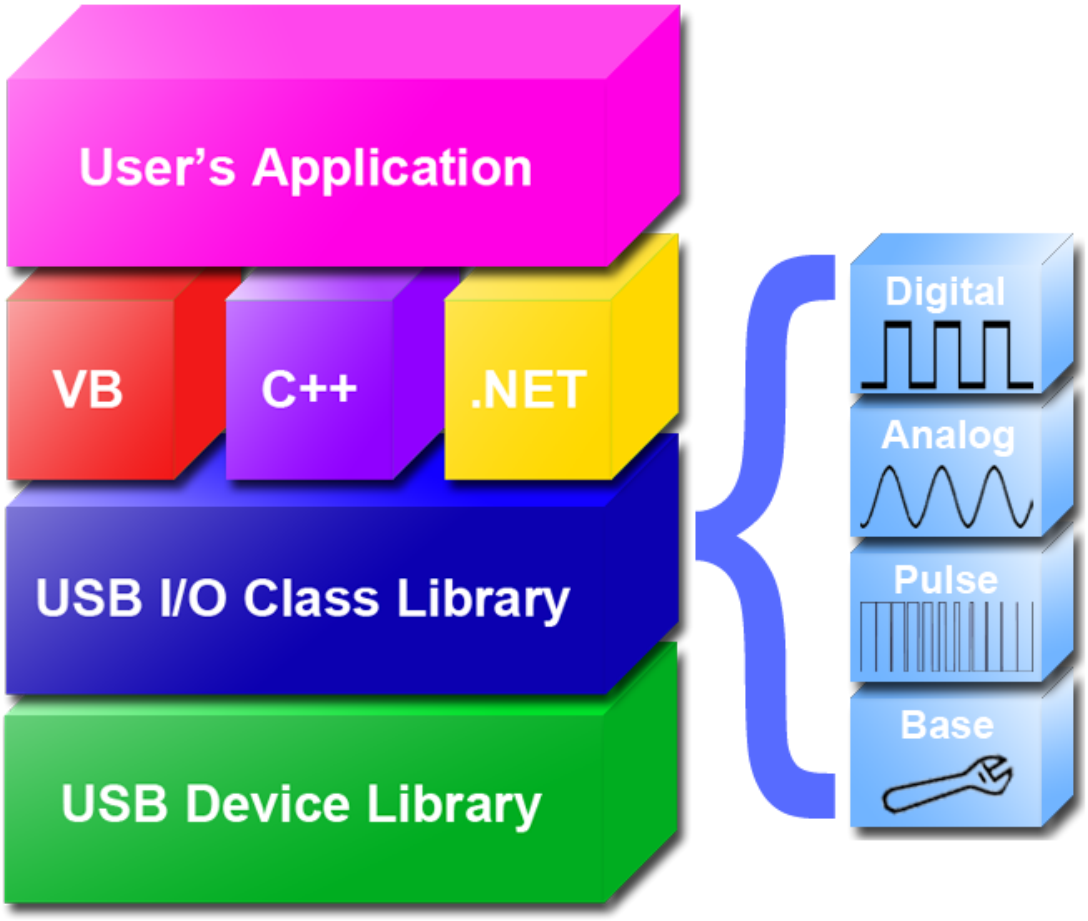
4.1 Hardware structure

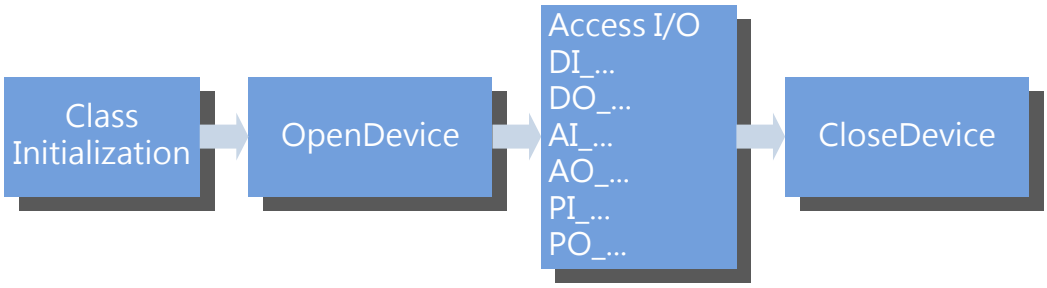
The ICP DAS USB I/O provides various types of input and output. The I/O is handled by embedded controller. The hardware structure is shown in figure 4-1 below.

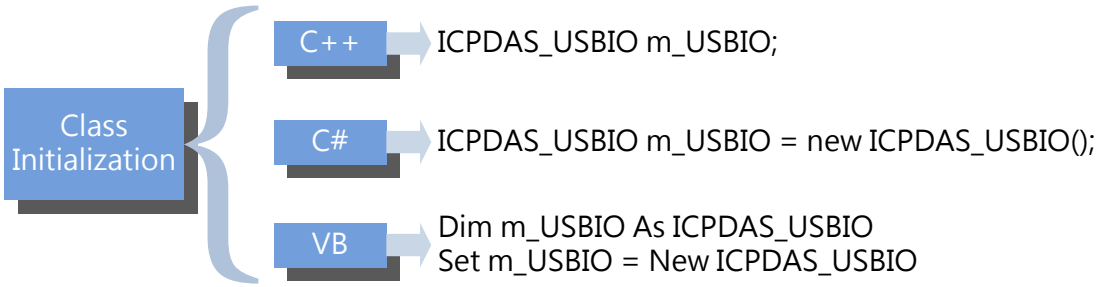


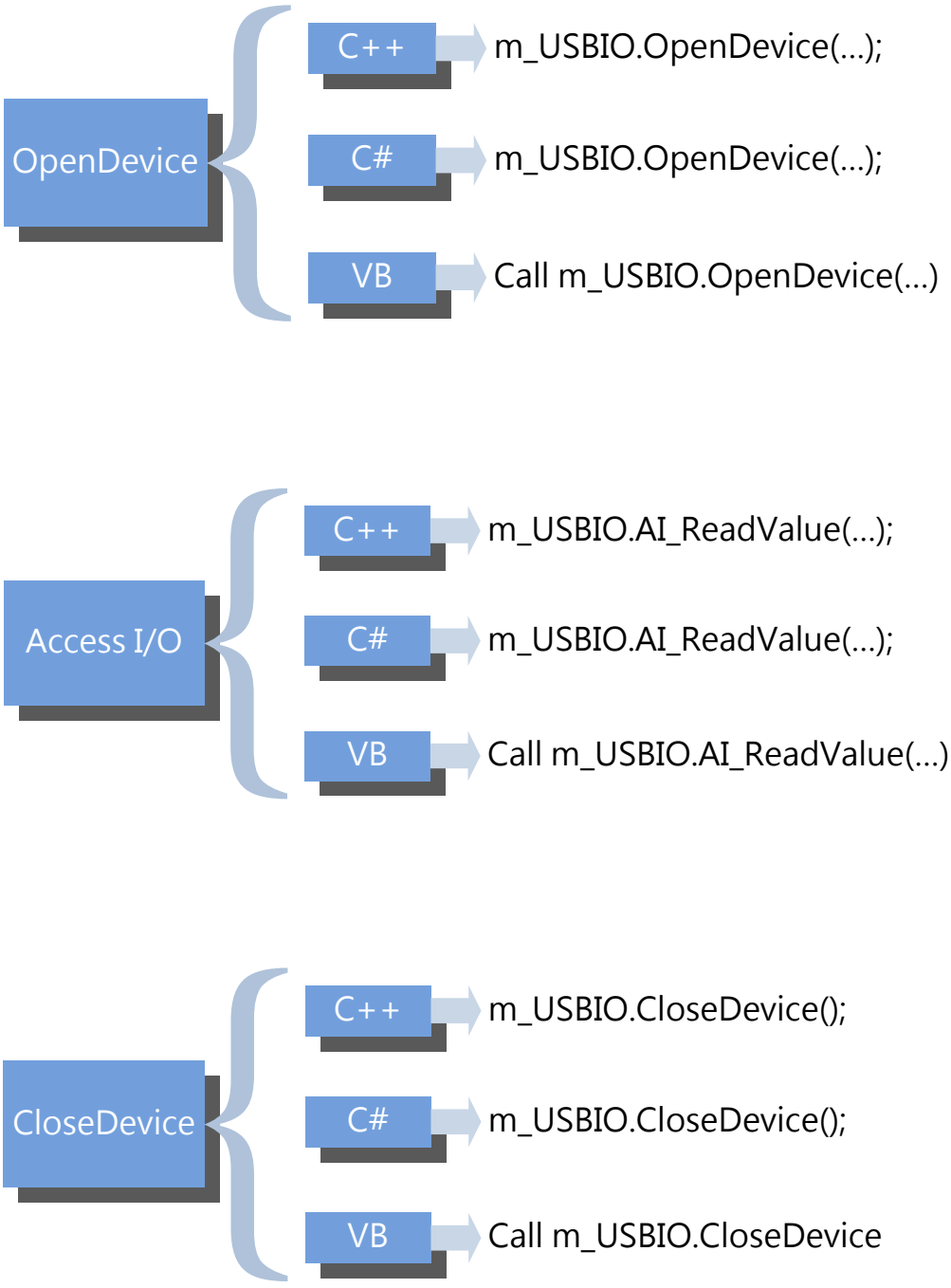
4.2 Software structure

In the programmer point of view, the ICP DAS provides a class library to user to develop project quickly and easily. The structure of the software is shown in figure 4-2. The methods of USB classes are divided into 4 groups, base, digital I/O, analog I/O and pulse I/O. The figures 4-3~4-7 show an overview to use ICP DAS USB I/O class library.


User's Application
VB
C++
.NET
USB I/O Class Library
USB Device Library
Digital
Analog
Pulse
Base


Class Initialization
OpenDevice
Access I/O
DI_...
DO_...
AI_...
AO_...
PI_...
PO_...
CloseDevice


Class Initialization
C++
ICPDAS_USBIO m_USBIO;
C#
ICPDAS_USBIO m_USBIO = new ICPDAS_USBIO();
VB
Dim m_USBIO As ICPDAS_USBIO
Set m_USBIO = New ICPDAS_USBIO


OpenDevice
C++
m_USBIO.OpenDevice(...);
C#
m_USBIO.OpenDevice(...);
VB
Call m_USBIO.OpenDevice(...)
Access I/O
C++
m_USBIO.AI_ReadValue(...);
C#
m_USBIO.AI_ReadValue(...);
VB
Call m_USBIO.AI_ReadValue(...)
CloseDevice
C++
m_USBIO.CloseDevice();
C#
m_USBIO.CloseDevice();
VB
Call m_USBIO.CloseDevice

5 ICP DAS USB Class Members

The members of the ICPDAS_USBIO class are divided into constructors, static and public methods. The constructors initialize and create the instance of the ICPDAS_USBIO. Static methods are the ways to identify or scan what USB modules are connected. Public methods are used to access USB modules. The following tables list the members of ICPDAS_USBIO class. The detail of these will be described in the following section.

5.1 Table of Constructors

Name	Description
ICPDAS_USBIO	Initializes a new instance of the ICPDAS_USBIO class.

5.2 Table of Static Methods

Name	Description
ListDevice	List all devices connected with local PC.
ScanDevice	Scan devices connected with local PC

5.3 Table of Public Methods

5.3.1 System

Name	Description
OpenDevice	List all devices connected with local PC.
CloseDevice	Scan devices connected with local PC.
SYNCDevice	Send a synchronization packet to clear software WDT.
SetCommTimeout	Set communication timeout.
GetCommTimeout	Get communication timeout.
SetAutoResetWDT	Enable / disable automatic reset of the WDT

5.3.2 Device

Name	Description
RefreshDeviceInfo	Refresh device information.
GetSoftWDTTimeout	Get software WDT timeout.
GetDeviceID	Get ID of the device.
GetFwVer	Get firmware version of the device.
GetDeviceNickName	Get nick name of the device.
GetDeviceSN	Get serial number of the device.
GetSupportIOMask	Get the mask of this device IO distribution.
GetDITotal	Get DI total channel of the device.
GetDOTotal	Get DO total channel of the device.
GetAITotal	Get AI total channel of the device.
GetAOTotal	Get AO total channel of the device.
GetPITotal	Get PI total channel of the device.
GetPOTotal	Get PO total channel of the device.
SetUserDefinedBoardID	Set board ID of this device.
SetDeviceNickName	Set nick name of this device.
SetSoftWDTTimeout	Set software WDT timeout.
LoadDefault	Load default setting.
StopBulk	Stop current bulk process.
RegisterEmergencyPktEventHandle	Register the callback function for emergency event sent from USBIO.

5.3.3 Digital Input

Name	Description
DI_GetDigitalFilterWidth	Digital Input function - Get DI Digital Filter Width
DI_GetDigitalValueInverse	Digital Input function - Get DI Value Inverse
DI_GetCntEdgeTrigger	Digital Input function - Get DI Counter Edge Trigger
DI_ReadValue	Digital Input function - Read DI Value
DI_ReadCounterValue	Digital Input function - Read DI Counter Value
DI_SetDigitalFilterWidth	Digital Input function - Set DI Digital Filter Width
DI_SetDigitalValueInverse	Digital Input function - Set DI Value Inverse
DI_SetCntEdgeTrigger	Digital Input function - Set DI Counter Edge Trigger
DI_WriteClearCounter	Digital Input function - Clear Specified Channel of DI

	Counter Value
DI_WriteClearCounters	Digital Input function - Clear DI Counter Value with Clear Mask

5.3.4 Digital Output

Name	Description
DO_GetPowerOnEnable	Digital Output function - Get Power-On Enable
DO_GetSafetyEnable	Digital Output function - Get Safety Enable
DO_GetSafetyValue	Digital Output function - Get Safety Value
DO_ReadValue	Digital Output function - Read DO Value
DO_SetPowerOnEnable	Digital Output function - Set Power-On Enable
DO_SetSafetyEnable	Digital Output function - Set Safety Enable
DO_SetSafetyValue	Digital Output function - Set Safety Value
DO_WriteValue	Digital Output function - Write DO Value

5.3.5 Analog Input

Name	Description
AI_GetTotalSupportType	Analog input function - Get total supported amount.
AI_GetSupportTypeCode	Analog input function - Get supported type code.
AI_GetTypeCode	Analog input function - Get type code.
AI_GetChCJCOffset	Analog input function - Get channel CJC offset.
AI_GetChEnable	Analog input function - Get channel enable/disable.
AI_GetFilterRejection	Analog input function - Get filter rejection.
AI_GetCJCOffset	Analog input function - Get CJC offset.
AI_GetCJCEnable	Analog input function - Get CJC enable.
AI_GetWireDetectEnable	Analog input function - Get wire detect enable.
AI_GetResolution	Analog input function - Get resolution.
AI_ReadValue	Analog input function - Read AI value in double word format. (Overload)
AI_ReadBulkValue	Analog input function - Read bulk AI value (Fast acquire functionality)
AI_ReadCJCValue	Analog input function - Get CJC value.
AI_SetTypeCode	Analog input function - Set type code for specific channel. (Overload)

AI_SetChCJCOffset	Analog input function - Set channel CJC offset for specific channel. (Overload)
AI_SetChEnable	Analog input function - Set channel enable/disable.
AI_SetFilterRejection	Analog input function - Set filter rejection.
AI_SetCJCOffset	Analog input function - Set CJC offset.
AI_SetCJCEnable	Analog input function - Set CJC enable.
AI_SetWireDetectEnable	Analog input function - Set wire detect enable.

5.3.6 Analog Output

Name	Description
AO_GetTotalSupportType	Analog output function - Get total supported amount.
AO_GetSupportTypeCode	Analog output function - Get supported type code.
AO_GetTypeCode	Analog output function - Get type code.
AO_GetChEnable	Analog output function - Get channel enable/disable.
AO_GetResolution	Analog output function - Get resolution.
AO_ReadExpValue	Analog output function - Read AO expect value in double word format.
AO_ReadCurValue	Analog output function - Read AO current value in double word format.
AO_GetPowerOnEnable	Analog output function – Get Power-On Enable.
AO_GetSafetyEnable	Analog output function - Get Safety Enable.
AO_GetPowerOnValue	Analog output function – Get Power-On value.
AO_GetSafetyValue	Analog output function – Get Safety value.
AO_SetTypeCode	Analog output function –Set type code for specific channel.
AO_SetChEnable	Analog output function - Set channel enable/disable.
AO_WriteValue	Analog output function – Write AO value in double word format.
AO_SetPowerOnEnable	Analog output function – Set Power-On Enable.
AO_SetSafetyEnable	Analog output function – Set Safety Enable.
AO_SetPowerOnValue	Analog output function – Set Power-On Value.
AO_SetSafetyValue	Analog output function – Set Safety Value.

5.3.7 Pulse Input

Name	Description
PI_GetTotalSupportType	Pulse input function - Get total supported amount.
PI_GetSupportTypeCode	Pulse input function - Get supported type code.
PI_GetTypeCode	Pulse input function - Get type code.
PI_GetTriggerMode	Pulse input function - Get trigger mode.

PI_GetLPFilterEnable	Pulse input function - Get low-pass filter enable.
PI_GetChIsolatedFlag	Pulse input function - Get channel isolated flag.
PI_GetLPFilterWidth	Pulse input function - Get low-pass filter width.
PI_ReadValue	Pulse input function - Read PI value.
PI_ReadCntValue	Pulse input function - Read the count value of counters
PI_ReadFreqValue	Pulse input function - Read the frequency value of counters
PI_ReadBulkValue	Pulse input function - Get bulk PI value (Fast acquire functionality)
PI_SetTypeCode	Pulse input function - Set type code for specific channel. (Overload)
PI_ClearChCount	Pulse input function - Clear channel count with clear mask.
PI_ClearSinglelChCount	Pulse input function - Clear single channel count.
PI_ClearChStatus	Pulse input function - Clear channel status with clear mask.
PI_ClearSinglelChStatus	Pulse input function - Clear single channel status.
PI_SetTriggerMode	Pulse input function - Set trigger mode. (Overload)
PI_SetChIsolatedFlag	Pulse input function - Set channel isolated flag. (Overload)
PI_SetLPFilterEnable	Pulse input function - Set low-pass filter enable. (Overload)
PI_SetLPFilterWidth	Pulse input function - Set low-pass filter width. (Overload)

5.3.8 Other

Name	Description
GetCurrentAccessObj	INTERNAL USE. DO NOT USE THIS METHOD.
SetNormalPktByteArray	INTERNAL USE. DO NOT USE THIS METHOD.
SetActivePktByteArray	INTERNAL USE. DO NOT USE THIS METHOD.
ClearActivePktBuffer	INTERNAL USE. DO NOT USE THIS METHOD.
GetActivePktByteArray	INTERNAL USE. DO NOT USE THIS METHOD.
SetNormalPktEvent	INTERNAL USE. DO NOT USE THIS METHOD.
IsDevMonitorThreadStop	INTERNAL USE. DO NOT USE THIS METHOD.
IsCommWithDevice	INTERNAL USE. DO NOT USE THIS METHOD.

GetLastCmdTime	INTERNAL USE. DO NOT USE THIS METHOD.

5.4 Constructors

5.4.1 ICPDAS_USBIO

Initialize a new instance of the ICPDAS_USBIO class.

Syntax

```
public ICPDAS_USBIO (
      void
)
```

Example

```
ICPDAS_USBIO m_usbIO;


m_usbIO = new ICPDAS_USBIO();
```

5.5 Static Methods

5.5.1 ListDevice

List all devices connected to local PC.

Syntax

```
public byte ListDevice (
    WORD *o_wDID,
    BYTE *o_byBID
)
```

Parameters

*o_wDID

[OUT] An array of device ID for all devices

*o_byBID

[OUT] An array of board ID for all devices

Return Value

Number of devices connected with PC

Example

```
BYTE byNumDevice, byBIDs[127];
WORD wDIDs[127];


byNumDevice = ICPDAS_USBIO.ListDevice(&wDIDs, &byBIDs);
```

5.5.2 ScanDevice

Scanning device connected to PC. This static method just refreshes the list of the ICP DAS USB series I/O modules, it is necessary to call ListDevice() to refresh new list.

Syntax

```
public int ScanDevice (
    void
)
```

Parameters

none

Return Value

Error code

Example

```
ICPDAS_USBIO.ScanDevice();
```

5.6 Public Methods

5.6.1 System

5.6.1.1 OpenDevice

Open USBIO with device ID and board ID. The device ID is defined by the header ICPDAS_USBIO.h or the enumeration in ICPDAS_USBIO.

Syntax

```
public int OpenDevice (
    WORD i_wUSBIO_DID,
    BYTE i_byUSBIO_BID
)
```

Parameters

i_wUSBIO_DID

[IN] Device ID for the specific device to open (Defined in ICPDAS_USBIO.h)

i_byUSBIO_BID

[IN] Board ID for the specific device to open

Return Value

Error code

Example

```
Int iErrCode;
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
iErrCode = m_usbIO.OpenDevice(USB2019, 1);
iErrCode = m_usbIO.CloseDevice();
```

5.6.1.2 CloseDevice

Close device and release resource.

Syntax

```
public int CloseDevice (
    void
)
```

Parameters

none

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;


m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
   // Some code accessing USB I/O
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.1.3 SYNCDevice

Send synchronization packet to I/O module.

Note 1: The synchronization will be handled by library automatically after calling OpenDevice, the synchronization will be closed after calling CloseDevice. User can call this API to send synchronization packet manually, and it will not stop the original synchronization.

Syntax

```
public int SYNCDevice (
    void
)
```

Parameters

none

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
   If(ERR_NO_ERR != (iErrCode = m_usbIO.SYNCDevice()))
      printf( "%d" , iErrCode);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.1.4 SetCommTimeout

Set the communication timeout between packet send and receive.

Note 1: The timeout value will affect communication. If the timeout is small, it means the communication is timeout after the value passed.

Note 2: The default value when first initial an ICP DAS USB I/O is 100ms.

Syntax

```
public int SetCommTimeout (
    DWORD i_dwCommTimeout
)
```

Parameters

i_dCommTimeout

[IN] The communication timeout in millisecond(ms)

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;


m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR != (iErrCode = m_usbIO.SetCommTimeout(1000)))
   printf( "%d" , iErrCode)
```

5.6.1.5 GetCommTimeout

Get the communication timeout between packet send and receive.

Note 1: The timeout value will affect communication. If the timeout is small, it means the communication is timeout after the value passed.

Note 2: The default value when first initial an ICP DAS USB I/O is 100ms.

Syntax

```
public int GetCommTimeout (
    DWORD* o_dwCommTimeout
)
```

Parameters

o_dCommTimeout

[OUT] The communication timeout in millisecond(ms)

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
DWORD o_dwCommTimeout;

m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.SetCommTimeout(1000)))
  if(ERR_NO_ERR != (iErrCode = m_usbIO. GetCommTimeout (&o_dwCommTimeout)))
    printf( "%d" , iErrCode);
  else
    printf( "%d\n" , o_dwCommTimeout);
```

5.6.1.6 SetAutoResetWDT

Enable / disable the handle of watchdog by library.

The library takes care of the watchdog automatically when first loaded. This advantage brings an easy way to access with USB modules. But in other side, sometimes users want to handle watchdog themselves. This API offers this functionality to disable the library to automatically handle watchdog.

NOTE1: The library will return to automatic when open device. This means users have to disable when open device.

Syntax

```
public int SetAutoResetWDT (
    BOOL i_bEnable
)
```

Parameters

i_bEnable

[IN] To enable / disable the library to automatically handle watchdog.

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;


m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
   If(ERR_NO_ERR != (iErrCode = m_usbIO.SetAutoResetWDT(FALSE)))
      printf( "%d" , iErrCode);
}
```

5.6.2 Device

5.6.2.1 RefreshDeviceInfo

Refresh all information of this device.

Note 1: The RefreshDeviceInfo() will be called automatically when open device.

Note 2: This function will take time to refresh information.

Syntax

```
public int RefreshDeviceInfo (
    void
)
```

Parameters

none

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;


m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
   if(ERR_NO_ERR != iErrCode = m_usbIO.RefreshDeviceInfo()))
      printf( "%d" , iErrCode)
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.2.2 GetSoftWDTTimeout

Get the software WDT timeout of I/O module.

Syntax

```
public int GetSoftWDTTimeout (
    DWORD *o_dwSoftWDTTimeout
)
```

Parameters

*o_dwSoftWDTTimeout

[OUT] The software WDT timeout in millisecond(ms)

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
DWORD o_dwSoftWDTTimeout;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
   if(ERR_NO_ERR != iErrCode = m_usbIO. GetSoftWDTTimeout (&o_dwSoftWDTTimeout)))
      printf( "%d" ,iErrCode);
   else
      printf( "%d\n" , o_dwCommTimeout);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.2.3 GetDeviceID

Get ID of the device.

Syntax

```
public int GetDeviceID (
    DWORD *o_dwDeviceID
)
```

Parameters

*o_dwDeviceID

[OUT] The device ID

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
DWORD o_dwDeviceID;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. GetDeviceID (&o_dwDeviceID)))
      printf( "%d" , iErrCode);
   else
      printf( "%d" , o_dwDeviceID);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.2.4 GetFwVer

Get firmware version of the device.

Syntax

```
public int GetDeviceID (
    WORD *o_ wFwVer
)
```

Parameters

*o_wFwVer

[OUT] The firmware version

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
WORD o_wFwVer;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. GetFwVer (&o_wFwVer)))
      printf( "%d" , iErrCode);
   else
      printf( "%d" ,o_wDwVer);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.2.5 GetDeviceNickName

Get nick name of the device.

Syntax

```
public int GetDeviceNickName (
    BYTE *o_byDeviceNickName
)
```

Parameters

*o_byDeviceNickName

[OUT] The byte array of the nick name of the device

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byDeviceNickName [USBIO_NICKNAME_LENGTH];

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. GetDeviceNickName (o_byDeviceNickName)))
      printf( "%d" , iErrCode);
   else
      printf( "%s" , o_byDeviceNickName);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.2.6 GetDeviceSN

Get serial number of the device.

Syntax

```
public int GetDeviceSN (
    BYTE *o_byDeviceSN
)
```

Parameters

*o_byDeviceSN

[OUT] The byte array of the serial number of the device

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byDeviceSN [USBIO_SN_LENGTH];

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. GetDeviceSN (o_ byDeviceSN)))
      printf( "%d" , iErrCode);
   else
      printf( "%s" , o_ byDeviceSN);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.2.7 GetSupportIOMask

Get the mask of this device IO distribution. Each bit of the mask indicates each supported IO type as shown in the following table.

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
N/A	N/A	PI	PO	AI	AO	DI	DO

This mask can help you to identify what types of IO are supported in the device.

Syntax

```
public int GetSupportIOMask (
    BYTE *o_bySupportIOMask
)
```

Parameters

*o_bySupportIOMask

[OUT] The support IO mask of the device

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_bySupportIOMask;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. GetSupportIOMask (&o_bySupportIOMask)))
      printf( "%d" , iErrCode);
   else
      printf( "0x%02x" ,o_bySupportIOMask);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.2.8 GetDITotal

Get DI total number of channels of the device.

Syntax

```
public int GetDITotal (
    BYTE *o_byDITotal
)
```

Parameters

*o_byDITotal

[OUT] The DI total number of channels

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byDITotal;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB20xx, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. GetDITotal (&o_byDITotal)))
      printf( "%d" , iErrCode);
   else
      printf( "%d" ,o_byDITotal);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.2.9 GetDOTotal

Get DO total number of channels of the device.

Syntax

```
public int GetDOTotal (
    BYTE *o_byDOTotal
)
```

Parameters

*o_byDOTotal

[OUT] The DO total number of channels

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byDOTotal;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB20xx, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. GetDOTotal (&o_byDOTotal)))
      printf( "%d" , iErrCode);
   else
      printf( "%d" ,o_byDOTotal);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.2.10 GetAITotal

Get AI total number of channels of the device.

Syntax

```
public int GetAITotal (
    BYTE *o_byAITotal
)
```

Parameters

*o_byAITotal

[OUT] The AI total number of channels

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byAITotal;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB20xx, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. GetAITotal (&o_byAITotal)))
      printf( "%d" , iErrCode);
   else
      printf( "%d" ,o_byAITotal);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.2.11 GetAOTotal

Get AO total number of channels of the device.

Syntax

```
public int GetAOTotal (
    BYTE *o_byAOTotal
)
```

Parameters

*o_byAOTotal

[OUT] The AO total number of channels

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byAOTotal;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB20xx, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. GetAOTotal (&o_byAOTotal)))
      printf( "%d" , iErrCode);
   else
      printf( "%d" ,o_byAOTotal);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.2.12 GetPITotal

Get PI total number of channels of the device.

Syntax

```
public int GetPITotal (
    BYTE *o_byPITotal
)
```

Parameters

*o_byPITotal

[OUT] The PI total number of channels

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byPITotal;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB20xx, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. GetPITotal (&o_byPITotal)))
      printf( "%d" , iErrCode);
   else
      printf( "%d" ,o_byPITotal);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.2.13 GetPOTotal

Get PO total number of channels of the device.

Syntax

```
public int GetPOTotal (
    BYTE *o_byPOTotal
)
```

Parameters

*o_byPOTotal

[OUT] The PO total number of channels

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byPOTotal;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB20xx, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. GetPOTotal (&o_byPOTotal)))
      printf( "%d" , iErrCode);
   else
      printf( "%d" ,o_byPOTotal);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.2.14 SetUserDefinedBoardID

Set board ID of this device. The valid value of the ID is from 16 to 127.

Syntax

```
public int SetUserDefinedBoardID (
    BYTE i_byBID
)
```

Parameters

i_byBID

[IN] The board ID to set

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB20xx, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. SetUserDefinedBoardID (123)))
      printf( "%d" , iErrCode);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.2.15 SetDeviceNickName

Set nick name of this device. The maximum number of the character of this device is 32.

Syntax

```
public int SetDeviceNickName (
    BYTE *i_byDeviceNickName
)
```

Parameters

*i_byDeviceNickName

[IN] The byte array of the nick name to set

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte byNickName[USBIO_NICKNAME_LENGTH];

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB20xx, 1)))
{
   sprintf(byNickName,  "Station 1-1-3" );
   if(ERR_NO_ERR != (iErrCode = m_usbIO. SetDeviceNickName (byNickName)))
      printf( "%d" , iErrCode);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.2.16 SetSoftWDTTimeout

Set the software WDT timeout.

The minimum value of timeout is 100ms, and maximum is 30 minutes.

Syntax

```
public int SetSoftWDTTimeout (
    DWORD i_dwSoftWDTTimeout
)
```

Parameters

i_dwSoftWDTTimeout

[IN] The software WDT timeout in millisecond(ms)

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;


m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB20xx, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. SetSoftWDTTimeout (1000)))
      printf( "%d" , iErrCode);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.2.17 LoadDefault

Load default setting.

Syntax

```
public int LoadDefault (
    void
)
```

Parameters

none

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB20xx, 1)))
{
   If(ERR_NO_ERR != (iErrCode = m_usbIO.LoadDefault ()))
      printf(  "%d" , iErrCode);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.2.18 StopBulk

Stop current bulk process.

Syntax

```
public int LoadDefault (
    void
)
```

Parameters

none

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;


m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB20xx, 1)))
{
   If(ERR_NO_ERR != (iErrCode = m_usbIO.StopBulk ()))
      printf(  "%d" , iErrCode);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.2.19 RegisterEmergencyPktEventHandle

Register the callback function for emergency event sent from USBIO.

When in callback operation, it will cause the performance in your callback function.

Please reduce execute time in this callback function.

Syntax

```
public int RegisterEmergencyPktEventHandle (
    OnEmergencyPktArriveEvent i_evtHandle
)
```

Parameters

i_evtHandle

[IN] The callback function for emergency event

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Bool m_bEmcyPktArrive;
Byte byEmcyPkt[USBIO_MAX_PACKET_LENGTH];

Void emcypkeEvt(Byte* byData, Byte byLen)
{
   m_ bEmcyPktArrive = true;
   memcpy(byEmcyPkt, byData, byLen);
}

m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB20xx, 1)))
{
   If(ERR_NO_ERR != (iErrCode = m_usbIO.RegisterEmergencyPktEventHandle (emcypkeEvt)))
      printf( "%d" , iErrCode);
   while(1)
   {
      // User' s application loop
      If(m_ bEmcyPktArrive)
      {
         // Handle emcy packet
      }
   }
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.3 Digital Input

5.6.3.1 DI_GetDigitalFilterWidth

Digital input function - Get DI Digital Filter Width. The digital filter width is used for filtering the noise or the glitch. The unit of the filter is 0.1 milli-second.

Syntax

```
public int DI_GetDigitalFilterWidth (
    WORD* o_wFilterWidth
)
```

Parameters

*o_wFilterWidth

[OUT] The digital filter width (The unit is 0.1 ms)

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
WORD o_wFilterWidth;
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2051, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.DI_GetDigitalFilterWidth (&o_wFilterWidth)))
   {
      printf( "%d" , iErrCode);
      bRet = false;
   }
   If(bRet)
   {
      printf( "%d\n" , o_wFilterWidth);
   }
}
```

5.6.3.2 DI_GetDigitalValueInverse

Digital input function - Get DI Value Inverse.

Inverse	Value
No	0
Yes	1

Syntax

```
public int DI_GetDigitalValueInverse (
    DWORD* o_dwInverse
)
```

Parameters

*o_dwInverse

[OUT] The inverse setting

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
DWORD o_dwInverse;
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2051, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.DI_GetDigitalValueInverse (&o_dwInverse)))
   {
      printf( "%d" , iErrCode);
      bRet = false;
   }
   If(bRet)
   {
      printf( "%d\n" , o_dwInverse);
   }
}
```

5.6.3.3 DI_GetCntEdgeTrigger

Digital input function - Get DI Counter Edge Trigger. This edge trigger is used for the counting operation of the counter.

Edge Trigger	Value
Falling	0
Rising	1

Syntax

```
public int DI_GetCntEdgeTrigger (
    DWORD* o_dwEdgeTrig
)
```

Parameters

*o_dwEdgeTrig

[OUT] The counter edge trigger setting

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
DWORD o_dwEdgeTrig;
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2051, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.DI_GetCntEdgeTrigger (&o_dwEdgeTrig)))
   {
      printf( "%d" , iErrCode);
      bRet = false;
   }
   If(bRet)
   {
      printf( "%d\n" , o_dwEdgeTrig);
   }
}
```

5.6.3.4 DI_ReadValue

Digital Input function - Read DI Value. The values of digital input channels in byte array format.

Syntax

```
public int DI_ReadValue (
    BYTE* o_byDIValue
)
```

Parameters

*o_byDIValue

[OUT] The byte arrays of the DI value

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
BYTE o_byDIValue[USBIO_DI_MAX_CHANNEL];
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2051, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.DI_ReadValue (o_byDIValue)))
   {
      printf( "%d" , iErrCode);
      bRet = false;
   }
   If(bRet)
   {
      for(iIdx = 0; iIdx < (USBIO_DI_MAX_CHANNEL >> 3); iIdx++)
         printf( "%d\n" , o_byDIValue[iIdx]);
   }
}
```

5.6.3.5 DI_ReadCounterValue

Digital Input function - Read DI Counter Value. The counting value of the digital input counter in word array format.

Syntax

```
public int DI_ReadCounterValue (
    DWORD* o_dwDICntValue
)
```

Parameters

*o_dwDICntValue

[OUT] The double-word arrays of the DI counter value

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
WORD o_dwDICntValue[USBIO_DI_MAX_CHANNEL];
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2051, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.DI_ReadCounterValue (o_dwDICntValue)))
   {
      printf( "%d" , iErrCode);
      bRet = false;
   }
   If(bRet)
   {
      for(iIdx = 0; iIdx < (USBIO_DI_MAX_CHANNEL >> 3); iIdx++)
         printf( "%d\n" , o_dwDICntValue[iIdx]);
   }
}
```

5.6.3.6 DI_SetDigitalFilterWidth

Digital input function - Set DI Digital Filter Width. Used for setting the filter width for the digital input. The unit of the filter width is 0.1 milli-second.

Syntax

```
public int DI_SetDigitalFilterWidth (
    WORD* i_wFilterWidth
)
```

Parameters

*i_wFilterWidth

[IN] The digital filter width (The unit is 0.1 ms)

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;


m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2051, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.DI_SetDigitalFilterWidth (10))
      printf( "%d" , iErrCode);
   m_usbIO.CloseDevice();
}
```

5.6.3.7 DI_SetDigitalValueInverse

Digital input function - Set DI Value Inverse.

Inverse	Value
No	0
Yes	1

Syntax

```
public int DI_SetDigitalValueInverse (
    DWORD i_dwInverse
)
```

Parameters

i_dwInverse

[IN] The inverse setting

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;


m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2051, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.DI_SetDigitalValueInverse (1))
      printf( "%d" , iErrCode);
   m_usbIO.CloseDevice();
}
```

5.6.3.8 DI_SetCntEdgeTrigger

Digital input function - Set DI Counter Edge Trigger.

Edge Trigger	Value
Falling	0
Rising	1

Syntax

```
public int DI_SetCntEdgeTrigger (
    DWORD* i_dwEdgeTrig
)
```

Parameters

*i_dwEdgeTrig

[IN] The counter edge trigger setting

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;


m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2051, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.DI_SetCntEdgeTrigger (1))
      printf( "%d" , iErrCode);
   m_usbIO.CloseDevice();
}
```

5.6.3.9 DI_WriteClearCounter

Digital input function - Clear specified channel of DI Counter Value.

Syntax

```
public int DI_WriteClearCounter (
    BYTE* i_byChToClr
)
```

Parameters

*i_byChToClr

[IN] The counter channel for clearing

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2051, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.DI_WriteClearCounter (1))
      printf( "%d" , iErrCode);
   m_usbIO.CloseDevice();
}
```

5.6.3.10 DI_WriteClearCounters

Digital input function - Clear DI Counter Value with Clear Mask.

Syntax

```
public int DI_WriteClearCounters (
    DWORD* i_dwCntClrMask
)
```

Parameters

*i_dwCntClrMask

[IN] The counter clear mask

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2051, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.DI_WriteClearCounters (0x0000FFFF))
      printf( "%d" , iErrCode);
   m_usbIO.CloseDevice();
}
```

5.6.4 Digital Output

5.6.4.1 DO_GetPowerOnEnable

Digital Output function - Get Power-On Enable

Power-On Enable	Value
Disable & Off	0
Enable & On	1

Syntax

```
public int DO_GetPowerOnEnable(
    BYTE* o_byPowerOnEnable
)
```

Parameters

*o_byPowerOnEnable

[OUT] The power-on enable mask. Each byte represents the power-on enable / disable configuration of each channel.

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byPowerOnEnable[USBIO_DO_MAX_CHANNEL];
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2064, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. DO_GetPowerOnEnable (&o_byPowerOnEnable)))
   {
      printf( "%d" , iErrCode);
      bRet = false;
   }
   If(bRet)
   {
      for(iIdx = 0; iIdx <USBIO_DO_MAX_CHANNEL; iIdx++)
         printf( "%02x\n" , o_byPowerOnEnable[iIdx]);
   }
}
```

5.6.4.2 DO_GetSafetyEnable

Digital Output function - Get Safety Enable. Each channel takes one bit.

Safety Enable	Value
Disable	0
Enable	1

Syntax

```
public int DO_GetSafetyEnable (
    BYTE* o_bySafetyEnables
)
```

Parameters

*o_bySafetyEnables

[OUT] The safety enable mask. Each bit of the mask represents the safety enable / disable configuration of each channel

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_bySafetyEnable[(USBIO_DO_MAX_CHANNEL + 7] / 8];
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2064, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.DO_GetSafetyEnable(&o_bySafetyEnables))
   {
      printf( "%d" , iErrCode);
      bRet = false;
   }
   If(bRet)
   {
      for(iIdx = 0; iIdx <((USBIO_DO_MAX_CHANNEL + 7] / 8); iIdx++)
         printf( "%02x\n" , o_bySafetyEnables[iIdx]);
   }
}
```

5.6.4.3 DO_GetSafetyValue

Digital Output function - Get Safety Value. Each channel takes one bit.

Safety Value	Value
Off	0
On	1

Syntax

```
public int DO_GetSafetyValue(
    BYTE* o_bySafetyValue
)
```

Parameters

*o_bySafetyValue

[OUT] The safety value. Each bit represents the safety value of each channel.

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_bySafetyValue[(USBIO_DO_MAX_CHANNEL + 7] / 8];
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2064, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.DO_GetSafetyValue(&o_bySafetyValue))
   {
      printf( "%d" , iErrCode);
      bRet = false;
   }
   If(bRet)
   {
      for(iIdx = 0; iIdx <((USBIO_DO_MAX_CHANNEL + 7] / 8); iIdx++)
         printf( "%02x\n" , o_bySafetyValue[iIdx]);
   }
}
```

5.6.4.4 DO_GetDigitalOutputInverse

Digital output function - Get DO Output Inverse.

Inverse	Value
No	0
Yes	1

Syntax

```
public int DO_GetDigitalOutputInverse (
    DWORD* o_dwInverse
)
```

Parameters

*o_dwInverse

[OUT] The inverse setting

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
DWORD o_dwInverse;
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2045, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. DO_GetDigitalOutputInverse (&o_dwInverse)))
   {
      printf( "%d" , iErrCode);
      bRet = false;
   }
   If(bRet)
   {
      printf( "%d\n" , o_dwInverse);
   }
}
```

5.6.4.5 DO_ReadValue

Digital Output function - Read DO Value

Syntax

```
public int DO_ReadValue(
    BYTE* o_byDOValue
)
```

Parameters

*o_byDOValue

[OUT] The DO value. Each bit represents the DO value of each channel.

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byDOValue[(USBIO_DO_MAX_CHANNEL + 7] / 8];
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2064, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.DO_ReadValue(&o_byDOValue))
   {
      printf( "%d" , iErrCode);
      bRet = false;
   }
   If(bRet)
   {
      for(iIdx = 0; iIdx <((USBIO_DO_MAX_CHANNEL + 7] / 8); iIdx++)
         printf( "%02x\n" , o_byDOValue[iIdx]);
   }
}
```

5.6.4.6 DO_SetPowerOnEnable

The class has two overload methods for configuring power on functionality. One provides specifying channel to set, another for all channel. These two overload methods are listed as following table and described in following section.

Name of Methods
DO_SetPowerOnEnable(BYTE i_byChToSet, BYTE i_byPowerOnEnable)
DO_SetPowerOnEnable(BYTE* i_byPowerOnEnables)

5.6.4.6.1 DO_SetPowerOnEnable (BYTE, BYTE)

Digital Output function - Set Power-On enable for specific channel. The value of the enable byte is listed below.

Power-On Enable	Value
Disable & Off	0
Enable & On	1

Syntax

```
public int DO_SetPowerOnEnable (
    BYTE i_byChToSet,
    BYTE i_byPowerOnEnable
)
```

Parameters

i_byChToSet

[IN] The specific channel to set

i_byTypeCode

[IN] The power-on enable for the specific channel

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;


m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2064, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.DO_SetPowerOnEnable(0, 0x1)))
      printf( "%d" , iErrCode);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.4.6.2 DO_SetPowerOnEnable (BYTE*)

Digital Output function - Set Power-On enable for all channel. The value of the enable byte is listed below.

Power-On Enable	Value
Disable & Off	0
Enable & On	1

Syntax

```
public int DO_SetPowerOnEnable (
    BYTE *i_byPowerOnEnables
)
```

Parameters

*i_byPowerOnEnables

[IN] The byte array of the power-on enable

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte i_byChPwrOn[USBIO_DO_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2064, 1)))
{
   For(iIdx = 0; iIdx < USBIO_DO_MAX_CHANNEL; iIdx)
      i_byChPwrOn[iIdx] = 0x1;
   if(ERR_NO_ERR != (iErrCode = m_usbIO.DO_SetPowerOnEnable(i_byChPwrOn)))
      printf( "%d" , iErrCode);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.4.7 DO_SetSafetyEnable

Digital Output function - Set Safety Enable. Each bit represents channel safety enable.

The value of the safety enable is shown below.

Safety Enable	Value
Disable	0
Enable	1

Syntax

```
public int DO_SetSafetyEnable(
    BYTE* i_bySafetyEnable
)
```

Parameters

*i_bySafetyEnable

[IN] The safety enable mask. Each bit represents the safety configuration of each channel.

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte i_bySafetyEnable[(USBIO_DO_MAX_CHANNEL + 7] / 8];
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2064, 1)))
{
   memset(i_bySafetyEnable, 0xFF, (USBIO_DO_MAX_CHANNEL + 7] / 8);
   if(ERR_NO_ERR != (iErrCode = m_usbIO.DO_SetSafetyEnable(i_bySafetyEnable))
      printf( "%d" , iErrCode);
   m_usbIO.CloseDevice();
}
```

5.6.4.8 DO_SetSafetyValue

Digital Output function - Set Safety Value. Each bit represents safety value. The value of the safety is shown below.

Safety Value	Value
Off	0
On	1

Syntax

```
public int DO_SetSafetyValue(
    BYTE* i_bySafetyValue
)
```

Parameters

*i_bySafetyValue

[IN] The safety value. Each bit represents the safety value of each channel.

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte i_bySafetyValue[(USBIO_DO_MAX_CHANNEL + 7] / 8];
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2064, 1)))
{
   memset(i_bySafetyValue, 0xFF, (USBIO_DO_MAX_CHANNEL + 7] / 8);
   if(ERR_NO_ERR != (iErrCode = m_usbIO.DO_SetSafetyValue(i_bySafetyValue))
      printf( "%d" , iErrCode);
   m_usbIO.CloseDevice();
}
```

5.6.4.9 DO_SetDigitalOutputInverse

Digital output function - Set DO Output Inverse.

Inverse	Value
No	0
Yes	1

Syntax

```
public int DO_SetDigitalOutputInverse (
    DWORD i_dwInverse
)
```

Parameters

i_dwInverse

[IN] The inverse setting

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2051, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. DO_SetDigitalOutputInverse (1))
      printf( "%d" , iErrCode);
   m_usbIO.CloseDevice();
}
```

5.6.4.10 DO_WriteValue

The class has two overload methods for writing DO value. One provides specifying channel to write, another for all channel. These two overload methods are listed as following table and described in following section.

Name of Methods
DO_WriteValue (BYTE i_byChannel, BYTE i_byValue)
DO_WriteValue (BYTE *i_byDOValue)

5.6.4.10.1 DO_WriteValue (BYTE, BYTE)

Digital Output function - Write DO Value to specific channel. The value of the digital output is shown below.

Value	Value
Off	0
On	1

Syntax

```
public int DO_WriteValue(
    BYTE i_byChannel
    BYTE i_byValue
)
```

Parameters

i_byChannel

[IN] The specific DO channel to be set.

i_byValue

[IN] The DO on / off bit.

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2064, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.DO_WriteValue(5, 1))
      printf( "%d" , iErrCode);
   m_usbIO.CloseDevice();
}
```

5.6.4.10.2 DO_WriteValue (BYTE *)

Digital Output function - Write DO Value. Each bit represents channel value. The value of the digital output is shown below.

Enable Bit	Value
Off	0
On	1

Syntax

```
public int DO_WriteValue(
    BYTE* i_byDOValue
)
```

Parameters

*i_byDOValue

[IN] The DO value. Each bit represents the digital output value of each channel.

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte i_byDOValue[(USBIO_DO_MAX_CHANNEL + 7] / 8];
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2064, 1)))
{
   memset(i_byDOValue, 0xFF, (USBIO_DO_MAX_CHANNEL + 7] / 8);
   if(ERR_NO_ERR != (iErrCode = m_usbIO.DO_WriteValue(i_byDOValue))
      printf( "%d" , iErrCode);
   m_usbIO.CloseDevice();
}
```

5.6.5 Analog Input

5.6.5.1 AI_GetTotalSupportType

Analog input function - Get total supported amount.

Syntax

```
public int AI_GetTotalSupportType (
    BYTE *o_byTotalSupportType
)
```

Parameters

*o_byTotalSupportType

[OUT] The number of total support type

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byTotalSupportType;
Byte o_bySupportTypeCode[USBIO_MAX_SUPPORT_TYPE];
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO. AI_GetTotalSupportType (&o_ byTotalSupportType)))
  {
    printf( "%d" , iErrCode);
    bRet = false;
  }
  if(ERR_NO_ERR != (iErrCode = m_usbIO. AI_GetSupportTypeCode (o_ bySupportTypeCode)))
  {
    printf( "%d" , iErrCode);
    bRet = false;
  }
  If(bRet)
  {
    printf( "%d\n" , o_byTotalSupportType);
    for(iIdx = 0; iIdx < o_byTotalSupportType; iIdx++)
      printf( "%02x\n" , o_bySupportTypeCode[iIdx]);
  }
}
```

5.6.5.2 AI_GetSupportTypeCode

Analog input function - Get supported type code Please refer to Appendix A.1 of user's manual to map AI channels input type.

Syntax

```
public int AI_GetTotalSupportType (
    BYTE *o_bySupportTypeCode
)
```

Parameters

*o_byTotalSupportType

[OUT] The number of total support type

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byTotalSupportType;
Byte o_bySupportTypeCode[USBIO_MAX_SUPPORT_TYPE];
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. AI_GetTotalSupportType (&o_ byTotalSupportType)))
   {
      printf( "%d" , iErrCode);
      bRet = false;
   }
   if(ERR_NO_ERR != (iErrCode = m_usbIO. AI_GetSupportTypeCode (o_ bySupportTypeCode)))
   {
      printf( "%d" , iErrCode);
      bRet = false;
   }
   If(bRet)
   {
      printf( "%d\n" , o_byTotalSupportType);
      for(iIdx = 0; iIdx < o_byTotalSupportType; iIdx++)
         printf( "%02x\n" , o_bySupportTypeCode[iIdx]);
   }
}
```

5.6.5.3 AI_GetTypeCode

Analog input function - Get type code Please refer to user's manual to map AI channels input type. The type code can reference to Appendix A.1.

Syntax

```
public int AI_GetTypeCode (
    BYTE *o_byTypeCode
)
```

Parameters

*o_byTypeCode

[OUT] The byte array of type code

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byTypeCode [USBIO_AI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO. AI_GetTypeCode (o_byTypeCode)))
    printf( "%d" , iErrCode);
  else
  {
    for(iIdx = 0; iIdx < USBIO_AI_MAX_CHANNEL; iIdx++)
      printf( "%02x\n" , o_byTypeCode[iIdx]);
  }
}
```

5.6.5.4 AI_GetChCJCOffset

Analog input function - Get channel CJC offset The valid range of offset is -40.96 ~ +40.95.

Syntax

```
public int AI_GetChCJCOffset (
    float *o_fChCJCOffset
)
```

Parameters

*o_ fChCJCOffset

[OUT] The float array of channel CJC offset

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
float o_fChCJCOffset [USBIO_AI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. AI_GetChCJCOffset (o_fChCJCOffset)))
      printf( "%d" , iErrCode);
   else
   {
      for(iIdx = 0; iIdx < USBIO_AI_MAX_CHANNEL; iIdx++)
         printf( "%.5f\n" , o_fChCJCOffset [iIdx]);
   }
}
```

5.6.5.5 AI_GetChEnable

Analog input function - Get channel enable/disable. Each byte indicates 8 channels enable/disable mask. EX: Byte0 -> Channel0 ~ 7

Syntax

```
public int AI_GetChCJCOffset (
    BYTE *o_byChEnable
)
```

Parameters

*o_byChEnable

[OUT] The byte array of channel enable/disable mask

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byChEnable [(USBIO_AI_MAX_CHANNEL + 7] / 8];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_GetChEnable (o_byChEnable)))
      printf( "%d" , iErrCode);
   else
   {
      for(iIdx = 0; iIdx < (USBIO_AI_MAX_CHANNEL + 7) / 8; iIdx++)
         printf( "%02x\n" , o_byChEnable [iIdx]);
   }
}
```

5.6.5.6 AI_GetFilterRejection

Analog input function - Get filter rejection.

Rejection Setting	Value
60Hz	0
50Hz	1

Syntax

```
public int AI_GetFilterRejection (
    BYTE *o_byFilterRejection
)
```

Parameters

*o_byFilterRejection

[OUT] The filter rejection

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byFilterRejection;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. AI_GetFilterRejection (&o_byFilterRejection)))
      printf( "%d" , iErrCode);
   else
      printf( "%d\n" , o_byFilterRejection);
}
```

5.6.5.7 AI_GetCJCOffset

Analog input function - Get CJC offset The valid range of offset is -40.96 ~ +40.95.

Syntax

```
public int AI_GetCJCOffset (
    float *o_fCJCOffset
)
```

Parameters

*o_fCJCOffset

[OUT] The CJC offset

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
float o_fCJCOffset;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. AI_GetCJCOffset (&o_fCJCOffset)))
      printf( "%d" , iErrCode);
   else
      printf( "%.5f\n" , o_fCJCOffset);
}
```

5.6.5.8 AI_GetCJCEnable

Analog input function - Get CJC enable.

Enable Setting	Value
Disable	0
Enable	1

Syntax

```
public int AI_GetCJCEnable (
    BYTE *o_byCJCEnable
)
```

Parameters

*o_byCJCEnable

[OUT] The CJC enable

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byCJCEnable;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_GetCJCEnable (&o_byCJCEnable)))
      printf( "%d" , iErrCode);
   else
      printf( "%d\n" , o_byCJCEnable);
}
```

5.6.5.9 AI_GetWireDetectEnable

Analog input function - Get wire detect enable.

Enable Setting	Value
Disable	0
Enable	1

Syntax

```
public int AI_GetWireDetectEnable (
    BYTE *o_byWireDetectEnable
)
```

Parameters

*o_byWireDetectEnable

[OUT] The wire detect enable

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byWireDetectEnable;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. AI_GetWireDetectEnable (&o_byWireDetectEnable)))
      printf( "%d" , iErrCode);
   else
      printf( "%d\n" , o_byWireDetectEnable);
}
```

5.6.5.10 AI_GetResolution

Analog input function - Get resolution. Each byte indicates each channel real resolution.

Syntax

```
public int AI_GetResolution (
    BYTE *o_byResolution
)
```

Parameters

*o_byResolution

[OUT] The byte array of resolution for each channel

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byResolution[USBIO_AI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO. AI_GetResolution (o_byResolution)))
    printf( "%d" , iErrCode);
  else
  {
    For(iIdx = 0; iIdx < USBIO_AI_MAX_CHANNEL; iIdx++)
      printf( "%d\n" , o_byResolution[iIdx]);
  }
}
```

5.6.5.11 AI_ReadValue

The class library provides 4 overload methods to read AI value. Two methods, the parameter in float format, will convert raw value to true inside the method. Others will return raw value without having conversion. The overview of these methods is as following table, and will describe in the following section.

Name of Methods
AI_ReadValue (DWORD* o_dwAIValue)
AI_ReadValue (DWORD* o_dwAIValue, BYTE* o_byAIChStatus)
AI_ReadValue (float* o_fAIValue)
AI_ReadValue (float* o_fAIValue, BYTE* o_byAIChStatus)

5.6.5.11.1 AI_ReadValue (DWORD *)

Analog input function - Read AI value in double word (digital) format.

In the digital format, the value represents the value from zero to full scale. Ex: For type -10V ~ +10V, the value 0x0 indicates -10V and 0xFFFF (16bit resolution) indicates +10V.

Please note that, when channel was not in good status, the reading value no longer represents zero to full scale. Different channel status follows the following rule:

- Channel Over
  The reading value represents a sign value X indicates how many value over full scale range. This value can be calculated by following formula:
  Assume current type is -10V ~ +10V with 16 bit resolution and reading value is 0x13E, then we can get the actual value Y is $\mathrm{Y} = \left(1 + \frac{0\mathrm{X13E}}{0\mathrm{xFFFF}}\right) \times \left(10 - (-10)\right) + (-10)$
- Channel Under
  The reading value represents a sign value X indicates how many value under zero scale range. This value can be calculated by following formula:
  Assume current type is -5V ~ +5V with 16 bit resolution and reading value is 0x53E, then we can get the actual value Y is $\mathrm{Y} = \left(0 - \frac{0\mathrm{X53E}}{0\mathrm{xFFFF}}\right) \times \left(5 - (-5)\right) + (-5)$
- Channel Open & Channel Close
  The reading value of these two statuses will be the full scale for channel open and zero scale for channel close.

The overload API for only reading AI value cannot detect the channel status. It only read the AI value but has the most efficiency.

Syntax

```
public int AI_ReadValue (
    DWORD *o_dwAIValue
)
```

Parameters

*o_dwAIValue

[OUT] The raw value of AI value

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
DWORD o_dwAIValue[USBIO_AI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. AI_ReadValue(o_dwAIValue)))
      printf( "%d" , iErrCode);
   else
   {
      For(iIdx = 0; iIdx < USBIO_AI_MAX_CHANNEL; iIdx++)
         printf( "0x%08x\n" , o_dwAIValue[iIdx]);
   }
}
```

5.6.5.11.2 AI_ReadValue (DWORD *, BYTE*)

Analog input function - Read AI value in double word (digital) format.

In the digital format, the value represents the value from zero to full scale. Ex: For type -10V ~ +10V, the value 0x0 indicates -10V and 0xFFFF (16bit resolution) indicates +10V.

Please note that, when channel was not in good status, the reading value no longer represents zero to full scale. Different channel status follows the following rule:

- Channel Over

  The reading value represents a sign value X indicates how many value over full scale range. This value can be calculated by following formula:

  Assume current type is -10V ~ +10V with 16 bit resolution and reading value is 0x13E, then we can get the actual value Y is $\mathrm{Y} = \left(1 + \frac{0\mathrm{X13E}}{0\mathrm{xFFFF}}\right) \times \left(10 - (-10)\right) + (-10)$

- Channel Under

  The reading value represents a sign value X indicates how many value under zero scale range. This value can be calculated by following formula:

  Assume current type is -5V ~ +5V with 16 bit resolution and reading value is 0x53E, then we can get the actual value Y is $\mathrm{Y} = \left(0 - \frac{0\mathrm{X53E}}{0\mathrm{xFFFF}}\right) \times \left(5 - (-5)\right) + (-5)$

- Channel Open & Channel Close

  The reading value of these two statuses will be the full scale for channel open and zero scale for channel close.

Syntax

```
public int AI_ReadValue (
    DWORD *o_dwAIValue
    BYTE* o_byAIChStatus
)
```

Parameters

*o_dwAIValue

[OUT] The raw value of AI value

*o_byAIChStatus

[OUT] The byte array of channel status

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
DWORD o_dwAIValue[USBIO_AI_MAX_CHANNEL];
Byte o_byAIChStatus[USBIO_AI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO. AI_ReadValue(o_dwAIValue, o_byAIChStatus)))
    printf( "%d" , iErrCode);
  else
  {
    For(iIdx = 0; iIdx < USBIO_AI_MAX_CHANNEL; iIdx++)
      printf( "0x%08x,  0x%02x\n" , o_dwAIValue[iIdx], o_byAIChStatus);
  }
}
```

5.6.5.11.3 AI_ReadValue (float *)

Analog input function - Read the real AI value without channel status.

The reading value is calculated, users no need to convert it to real value for current input type. Ex: The reading value is 1.316 in -2.5 ~ +2.5V, the input signal is 1.316V.

Syntax

```
public int AI_ReadValue (
    float *o_fAIValue
)
```

Parameters

*o_fAIValue

[OUT] The true value of AI value

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
float o_fAIValue[USBIO_AI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. AI_ReadValue(o_fAIValue)))
      printf( "%d" , iErrCode);
   else
   {
      For(iIdx = 0; iIdx < USBIO_AI_MAX_CHANNEL; iIdx++)
         printf( "%.5f\n" , o_dwAIValue[iIdx]);
   }
}
```

5.6.5.11.4 AI_ReadValue (float *, BYTE*)

Analog input function - Read the real AI value with channel status.

Syntax

```
public int AI_ReadValue (
    float *o_fAIValue
    BYTE* o_byAIChStatus
)
```

Parameters

*o_fAIValue

[OUT] The true value of AI value

*o_byAIChStatus

[OUT] The byte array of channel status

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
float o_fAIValue[USBIO_AI_MAX_CHANNEL];
Byte o_byAIChStatus[USBIO_AI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. AI_ReadValue(o_fAIValue, o_byAIChStatus)))
      printf( "%d" , iErrCode);
   else
   {
      For(iIdx = 0; iIdx < USBIO_AI_MAX_CHANNEL; iIdx++)
         printf( "%.5f,   0x%02x\n" , o_fAIValue[iIdx], o_byAIChStatus);
   }
}
```

5.6.5.12 AI_ReadBulkValue

Analog input function – Trigger reading bulk AI value (Fast acquire functionality).

When in callback operation, it will cause the performance in your callback function. Please reduce execute time in this callback function.

The detail of operation is described as follow. When call this API, the AI module operation will be changed from normal to fast acquire mode. In fast acquire mode, AI module follow the parameter of API setting to acquire data.

The API has block and non-block operation. In block operation, user’ s application needs to wait until API finishing all procedure. In contrast with block mode, non-block provides a flexible way for user. In non-block operation, user’ s application can proceed to own other code. To enable non-block operation, it is important to declare a callback function and pass it through last parameter. For block operation, just pass a NULL definition in last parameter.

Due to the USB 2.0 Full-speed transfer rate capability, the maximum sample rate is 10 KHz.

Syntax

```
public int AI_ReadBulkValue (
    BYTE i_byStartCh,
    BYTE i_byChTotal,
    DWORD i_dwSampleWidth,
    Float i_fSampleRate,
    DWORD i_dwBufferWidth,
    DWORD *o_dwDataBuffer,
    OnBulkValueFinishEvent i_CBFunc
)
```

Parameters

i_byStartCh

[IN] The starting acquire channel

i_byChTotal

[IN] The total channels to acquire

i_dwSampleWidth

[IN] The sampling width (ms)

i_fSampleRate

[IN] The sampling rate (Hz). 10KHz maximum.

i_dwBufferWidth

[IN] The width of the buffer for single channel

*o_dwDataBuffer

[OUT] The 2-dimension buffer array to store

i_CBFunc

[IN] Block operation – NULL

[IN] Non-block operation - The address of callback function.

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
// To read 0~1 channel for 100ms in 5 KHz sample rate each channel in non-block operation
// So we have the following variable declaration
#define SampleRate 5000
#define BufferWidth 500; // 5000(Hz) * 0.1(100ms)
DWORD m_dwBuffer[2][BufferWidth];

Void BulkFinishCallback(DWORD dwCount)
{
   // Callback function to handle data
}

Int main()
{
   m_usbIO = new ICPDAS_USBIO();
   if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
   {
      if(ERR_NO_ERR != (iErrCode = m_usbIO. AI_ReadBulkValue(0,
                                                2,
                                                100,
                                                SampleRate,
                                                BufferWidth
                                                m_dwBuffer,
                                                BulkFinishCallback)))
         printf( "%d" , iErrCode);

      while(1) {Sleep(1);}
   }
}
```

5.6.5.13 AI_ReadCJCValue

Analog input function - Read the current CJC value on the module.

Syntax

```
public int AI_ReadCJCValue (
    float *o_fCJCValue
)
```

Parameters

*o_fCJCValue

[OUT] The CJC value

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
float o_fCJCValue;
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_ReadCJCValue(o_fCJCValue)))
      printf( "%d" , iErrCode);
   else
      printf( "%.5f\n" , o_fCJCValue);
}
```

5.6.5.14 AI_SetTypeCode

The class has two overload methods for setting type code. One provides specifying channel to set, another for all channel. Please refer to user's manual for analog input type code. These two overload methods are listed as following table and described in following section. The corresponding type code can be found in Appendix A.1.

Name of Methods
AI_SetTypeCode (BYTE i_byChToSet, BYTE i_byTypeCode)
AI_SetTypeCode (BYTE *i_byTypeCodes)

5.6.5.14.1 AI_SetTypeCode (BYTE, BYTE)

Analog input function - Set type code for specific channel. The type code can reference to Appendix A.1.

Syntax

```
public int AI_SetTypeCode (
    BYTE i_byChToSet,
    BYTE i_byTypeCode
)
```

Parameters

i_byChToSet

[IN] The specific channel to set

i_byTypeCode

[IN] The type code for the specific channel

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;


m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_SetTypeCode(0, 0x10)))
      printf( "%d" , iErrCode);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.5.14.2 AI_SetTypeCode (BYTE*)

Analog input function - Set type code for all channels. The type code can reference to Appendix A.1.

Syntax

```
public int AI_SetTypeCode (
    BYTE *i_byTypeCodes
)
```

Parameters

*i_byTypeCodes

[IN] The byte array of type code to set

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte m_byChTypeCode[USBIO_AI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
   For(iIdx = 0; iIdx < USBIO_AI_MAX_CHANNEL; iIdx)
      m_byChTypeCode[iIdx] = 0x10;
   if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_SetTypeCode(m_byChTypeCode)))
      printf( "%d" , iErrCode);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.5.15 AI_SetChCJCOffset

The class has two overload methods for setting channel CJC offset. One provides specifying channel to set, another for all channel. The valid range of offset is -40.96 ~ +40.95. These two overload methods are listed as following table and described in following section.

Name of Methods
AI_SetChCJCOffset (BYTE i_byChToSet, float i_fChCJCOffset)
AI_SetChCJCOffset (float *i_fChCJCOffsets)

5.6.5.15.1 AI_SetChCJCOffset (BYTE, float)

Analog input function - Set channel CJC offset for specific channel.

Syntax

```
public int AI_SetTypeCode (
    BYTE i_byChToSet,
    float i_fChCJCOffset
)
```

Parameters

i_byChToSet

[IN] The specific channel to set

i_fChCJCOffset

[IN] The CJC offset for the specific channel

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_SetChCJCOffset(0, 1.354)))
      printf( "%d" , iErrCode);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.5.15.2 AI_SetChCJCOffset (float*)

Analog input function - Set channel CJC offset for specific channel.

Syntax

```
public int AI_SetTypeCode (
    float* i_fChCJCOffset
)
```

Parameters

*i_fChCJCOffset

[IN] The float array of channel CJC offset to set

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
float m_fChCJCOffset[USBIO_AI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
   For(iIdx = 0; iIdx < USBIO_AI_MAX_CHANNEL; iIdx)
      m_fChCJCOffset[iIdx] = 1.358;
   if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_SetChCJCOffset(m_fChCJCOffset)))
      printf( "%d" , iErrCode);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.5.16 AI_SetChEnable

Analog input function - Set channel enable/disable. Each byte indicates 8 channels enable/disable mask. Ex: Byte0 -> Channel0 ~ 7

Syntax

```
public int AI_SetChEnable (
    BYTE *i_byChEnable
)
```

Parameters

*i_byChEnable

[IN] The byte array of channel enable/disable mask

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte m_byChEnable[(USBIO_AI_MAX_CHANNEL + 7) / 8];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
   For(iIdx = 0; iIdx <( USBIO_AI_MAX_CHANNEL + 7) / 8; iIdx)
      m_byChEnable [iIdx] = 0x5A;
   if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_SetChEnable(m_byChEnable)))
      printf( "%d" , iErrCode);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.5.17 AI_SetFilterRejection

Analog input function - Set filter rejection.

Rejection Setting	Value
60Hz	0
50Hz	1

Syntax

```
public int AI_SetFilterRejection (
    BYTE i_byFilterRejection
)
```

Parameters

i_byFilterRejection

[IN] The filter rejection

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;


m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_SetFilterRejection(0)))
      printf( "%d" , iErrCode);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.5.18 AI_SetCJCOffset

Analog input function - Set CJC offset. The valid range of offset is -40.96 ~ +40.95.

Syntax

```
public int AI_SetCJCOffset (
    float i_fCJCOffset
)
```

Parameters

i_fCJCOffset

[IN] The CJC offset

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_SetCJCOffset(-20.81)))
      printf( "%d" , iErrCode);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.5.19 AI_SetCJCEnable

Analog input function - Set CJC enable.

Enable Setting	Value
Disable	0
Enable	1

Syntax

```
public int AI_SetCJCOffset (
    BYTE i_byCJCEnable
)
```

Parameters

i_byCJCEnable

[IN] The CJC enable

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;


m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_SetCJCEnable(1)))
      printf( "%d" , iErrCode);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.5.20 AI_SetWireDetectEnable

Analog input function - Set wire detect enable.

Enable Setting	Value
Disable	0
Enable	1

Syntax

```
public int AI_SetCJCOffset (
    BYTE i_byWireDetectEnable
)
```

Parameters

i_byWireDetectEnable

[IN] The wire detect enable

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2019, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_SetWireDetectEnable(0)))
      printf( "%d" , iErrCode);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.6 Analog Output

5.6.6.1 AO_GetTotalSupportType

Analog output function - Get total supported amount.

Syntax

```
public int AO_GetTotalSupportType (
    BYTE *o_byTotalSupportType
)
```

Parameters

*o_byTotalSupportType

[OUT] The number of total support type

Return Value

Error code

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
BYTE o_byTotalSupportType;
BYTE o_bySupportTypeCode[USBIO_MAX_SUPPORT_TYPE];
int iIdx;
bool bRet = true;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{
	if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetTotalSupportType (&o_byTotalSupportType)))
	{
		printf(" %d", iErrCode);
		bRet = false;
	}
	if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetSupportTypeCode (o_bySupportTypeCode)))
	{
		printf("%d", iErrCode);
		bRet = false;
	}
	if(bRet)
	{
		printf("%d\n", o_byTotalSupportType);
		for(iIdx = 0; iIdx < o_byTotalSupportType; iIdx++)
			printf("%02x\n", o_bySupportTypeCode[iIdx]);
	}
}
return iErrCode;
```

5.6.6.2 AO_GetSupportTypeCode

Analog output function - Get supported type code Please refer to Appendix A.2 of user's manual to map AO channels output type.

Syntax

```
public int AO_GetTotalSupportType (
    BYTE *o_bySupportTypeCode
)
```

Parameters

*o_byTotalSupportType

[OUT] The number of total support type

Return Value

Error code

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
BYTE o_byTotalSupportType;
BYTE o_bySupportTypeCode[USBIO_MAX_SUPPORT_TYPE];
int iIdx;
bool bRet = true;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{
	if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetTotalSupportType (&o_byTotalSupportType)))
	{
		printf(" %d", iErrCode);
		bRet = false;
	}
	if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetSupportTypeCode (o_bySupportTypeCode)))
	{
		printf("%d", iErrCode);
		bRet = false;
	}
	if(bRet)
	{
		printf("%d\n", o_byTotalSupportType);
		for(iIdx = 0; iIdx < o_byTotalSupportType; iIdx++)
			printf("%02x\n", o_bySupportTypeCode[iIdx]);
	}
}
return iErrCode;
```

5.6.6.3 AO_GetTypeCode

Analog output function - Get type code Please refer to user's manual to map AO channels input type. The type code can reference to Appendix A.2.

Syntax

```
public int AO_GetTypeCode (
    BYTE *o_byTypeCode
)
```

Parameters

*o_byTypeCode

[OUT] The byte array of type code

Return Value

Error code

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
BYTE o_byTypeCode [USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. AO_GetTypeCode (o_byTypeCode)))
     printf("%d", iErrCode);
   else
   {
      for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
      printf("0x%02x\n", o_byTypeCode[iIdx]);
   }
}
return iErrCode;
```

5.6.6.4 AO_GetChEnable

Analog output function - Get channel enable/disable. Each byte indicates 8 channels enable/disable mask. EX: Byte0 -> Channel0 ~ 7

Syntax

```
public int AO_GetChEnable (
    BYTE *o_byChEnable
)
```

Parameters

*o_byChEnable

[OUT] The byte array of channel enable/disable mask

Return Value

Error code

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
BYTE o_byChEnable [(USBIO_AO_MAX_CHANNEL + 7) / 8];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetChEnable (o_byChEnable)))
     printf("%d", iErrCode);
   else
   {
      for(iIdx = 0; iIdx < (USBIO_AO_MAX_CHANNEL + 7) / 8; iIdx++)
      printf("0x%02x\n", o_byChEnable [iIdx]);
   }
}
return iErrCode;
```

5.6.6.5 AO_GetResolution

Analog output function - Get resolution. Each byte indicates each channel real resolution.

Syntax

```
public int AO_GetResolution (
    BYTE *o_byResolution
)
```

Parameters

*o_byResolution

[OUT] The byte array of resolution for each channel

Return Value

Error code

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
BYTE o_byResolution[USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. AO_GetResolution (o_byResolution)))
   printf("%d", iErrCode);
   else
   {
    for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
      printf("%d\n", o_byResolution[iIdx]);
   }
}
return iErrCode;
```

5.6.6.6 AO_ ReadExpValue

The class library provides 2 overload methods to read AO expect value. One method, the parameter in float format, will convert raw value to true inside the method. The other will return raw value without having conversion. The overview of these methods is as following table, and will describe in the following section.

Name of Methods
AO_ReadExpValue (DWORD* o_dwAOExpValue)
AO_ReadExpValue (float* o_fAOExpValue)

5.6.6.6.1 AO_ReadExpValue (DWORD *)

Analog output function - Read AO expect value in double word (digital) format.

In the digital format, the value represents the value from zero to full scale. Ex: For type -10V ~ +10V, the value 0x0 indicates -10V and 0xFFFF (16bit resolution) indicates +10V.

Please note that, when channel was not in good status, the reading value no longer represents zero to full scale. Different channel status follows the following rule:

- Channel Over
  The reading value represents a sign value X indicates how many value over full scale range. This value can be calculated by following formula:
  Assume current type is -10V ~ +10V with 16 bit resolution and reading value is 0x13E, then we can get the actual value Y is $\mathrm{Y} = \left(1 + \frac{0\mathrm{X13E}}{0\mathrm{xFFFF}}\right) \times \left(10 - (-10)\right) + (-10)$
- Channel Under
  The reading value represents a sign value X indicates how many value under zero scale range. This value can be calculated by following formula:
  Assume current type is -5V ~ +5V with 16 bit resolution and reading value is 0x53E, then we can get the actual value Y is $\mathrm{Y} = \left(0 - \frac{0\mathrm{X53E}}{0\mathrm{xFFFF}}\right) \times \left(5 - (-5)\right) + (-5)$
- Channel Open & Channel Close
  The reading value of these two statuses will be the full scale for channel open and zero scale for channel close.

The overload API for only reading AO value cannot detect the channel status. It only read the AO value but has the most efficiency.

Syntax

```
public int AO_ReadExpValue (
    DWORD * o_dwAOExpValue
)
```

Parameters

* o_dwAOExpValue

  [OUT] The raw value of AO expect value

Return Value

Error code

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
DWORD o_dwAOExpValue[USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{
if(ERR_NO_ERR != (iErrCode = m_usbIO. AO_ReadExpValue(o_dwAOExpValue)))
   printf("%d", iErrCode);
else
{
   for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
   printf("0x%04x\n", o_dwAOExpValue[iIdx]);
}
}
return iErrCode;
```

5.6.6.6.2 AO_ReadExpValue (float *)

Analog output function - Read the real AO expect value.

The reading value is calculated, users no need to convert it to real value for expect input type. Ex: The reading value is 1.316 in -2.5 ~ +2.5V, the input signal is 1.316V.

Syntax

```
public int AO_ReadExpValue (
    float * o_fAOExpValue
)
```

Parameters

* o_fAOExpValue

  [OUT] The true value of AO expect value

Return Value

Error code

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
float o_fAOExpValue[USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{
if(ERR_NO_ERR != (iErrCode = m_usbIO. AO_ReadExpValue(o_fAOExpValue)))
   printf("%d", iErrCode);
else
{
   for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
   printf("%04f\n", o_fAOExpValue[iIdx]);
}
} return iErrCode;
```

5.6.6.7 AO_ ReadCurValue

The class library provides 2 overload methods to read AO current value. One method, the parameter in float format, will convert raw value to true inside the method. The other will return raw value without having conversion. The overview of these methods is as following table, and will describe in the following section.

Name of Methods
AO_ReadCurValue (DWORD* o_dwAOCurValue)
AO_ReadCurValue (float* o_fAOCurValue)

5.6.6.7.1 AO_ReadCurValue (DWORD *)

Analog output function - Read AO current value in double word (digital) format.

In the digital format, the value represents the value from zero to full scale. Ex: For type -10V ~ +10V, the value 0x0 indicates -10V and 0xFFFF (16bit resolution) indicates +10V.

Please note that, when channel was not in good status, the reading value no longer represents zero to full scale. Different channel status follows the following rule:

- Channel Over
  The reading value represents a sign value X indicates how many value over full scale range. This value can be calculated by following formula:
  Assume current type is -10V ~ +10V with 16 bit resolution and reading value is 0x13E, then we can get the actual value Y is $\mathrm{Y} = \left(1 + \frac{0\mathrm{X13E}}{0\mathrm{xFFFF}}\right) \times \left(10 - (-10)\right) + (-10)$
- Channel Under
  The reading value represents a sign value X indicates how many value under zero scale range. This value can be calculated by following formula:
  Assume current type is -5V ~ +5V with 16 bit resolution and reading value is 0x53E, then we can get the actual value Y is $\mathrm{Y} = \left(0 - \frac{0\mathrm{X53E}}{0\mathrm{xFFFF}}\right) \times \left(5 - (-5)\right) + (-5)$
- Channel Open & Channel Close
  The reading value of these two statuses will be the full scale for channel open and zero scale for channel close.

The overload API for only reading AO value cannot detect the channel status. It only read the AO value but has the most efficiency.

Syntax

```
public int AO_ReadCurValue (
    DWORD * o_dwAOCurValue
)
```

Parameters

* o_dwAOCurValue

  [OUT] The raw value of AO current value

Return Value

Error code

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
DWORD o_dwAOCurValue[USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{
if(ERR_NO_ERR != (iErrCode = m_usbIO. AO_ReadCuValue(o_dwAOCurValue)))
   printf("%d", iErrCode);
else
{
   for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
   printf("0x%04x\n", o_dwAOCurValue[iIdx]);
}
}
return iErrCode;
```

5.6.6.7.2 AO_ReadCurValue (float *)

Analog output function - Read the real AO current value.

The reading value is calculated, users no need to convert it to real value for current input type. Ex: The reading value is 1.316 in -2.5 ~ +2.5V, the input signal is 1.316V.

Syntax

```
public int AO_ReadCurValue (
    float * o_fAOCurValue
)
```

Parameters

* o_fAOCurValue

  [OUT] The true value of AO current value

Return Value

Error code

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
float o_fAOCurValue[USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{
if(ERR_NO_ERR != (iErrCode = m_usbIO. AO_ReadExpValue(o_fAOCurValue)))
   printf("%d", iErrCode);
else
{
   for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
   printf("%04f\n", o_fAOCurValue[iIdx]);
}
} return iErrCode;
```

5.6.6.8 AO_GetPowerOnEnable

Analog output function - Get Power-On Enable. Each channel takes one byte.

Syntax

```
public int AO_GetPowerOnEnable (
    BYTE *o_byPowerOnEnable
)
```

Parameters

*o_byPowerOnEnable

[OUT] The byte array of channel enable/disable mask

Return Value

Error code

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
BYTE o_byPwrOnEnable [USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetPowerOnEnable (o_byPwrOnEnable)))
     printf("%d", iErrCode);
else
{
   for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
      printf("0x%02x\n", o_byPwrOnEnable [iIdx]);
}
}
return iErrCode;
```

5.6.6.9 AO_GetSafetyEnable

Analog output function - Get Safety Enable. Each byte indicates 8 channels enable/disable mask. EX: Byte0 -> Channel0 ~ 7

Syntax

```
public int AO_GetSafetyEnable (
    BYTE *o_bySafetyEnable
)
```

Parameters

*o_bySafetyEnable

[OUT] The byte array of Safety enable/disable mask

Return Value

Error code

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
BYTE o_bySafetyEnable [(USBIO_AO_MAX_CHANNEL + 7) / 8];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetSafetyEnable (o_bySafetyEnable)))
     printf("%d", iErrCode);
else
{
   for(iIdx = 0; iIdx < (USBIO_AO_MAX_CHANNEL + 7) / 8; iIdx++)
      printf("0x%02x\n", o_bySafetyEnable [iIdx]);
}
}
return iErrCode;
```

5.6.6.10 AO_GetPowerOnValue

The class library provides 2 overload methods to read AO Power-On Value. One method, the parameter in float format, will convert raw value to true inside the method. The other will return raw value without having conversion. The overview of these methods is as following table, and will describe in the following section.

Name of Methods
AO_ GetPowerOnValue (DWORD* o_ dwPwrOnValue)
AO_ GetPowerOnValue (float* o_ fPwrOnValue)

5.6.6.10.1 AO_GetPowerOnValue(DWORD*)

Analog output function - Get Power-On Value.Each channel takes one unit of DWORD array.

Syntax

```
public int AO_GetPowerOnValue (
    DWORD* o_dwPwrOnValue
)
```

Parameters

*o_ dwPwrOnValue

[OUT] The DWORD array of Power-On Value

Return Value

Error code

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
DWORD o_dwPwrOnValue [USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{
if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetPowerOnValue (o_dwPwrOnValue)))
    printf("%d", iErrCode);
else
{
for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
      printf("0x%04x\n", o_dwPwrOnValue[iIdx]);
}
}
return iErrCode;
```

5.6.6.10.2 AO_GetPowerOnValue(float*)

Analog output function - Get Power-On Value.Each channel takes one unit of float array.

Syntax

```
public int AO_GetPowerOnValue (
    float* o_fPwrOnValue
)
```

Parameters

*o_ fPwrOnValue

[OUT] The float array of Power-On Value

Return Value

Error code

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
float o_fPwrOnValue [USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{
if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetPowerOnValue (o_ fPwrOnValue)))
    printf("%d", iErrCode);
else
{
for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
      printf("%04f\n", o_ fPwrOnValue [iIdx]);
}
}
return iErrCode;
```

5.6.6.11 AO_GetSafetyValue

The class library provides 2 overload methods to read AO Safety Value. One method, the parameter in float format, will convert raw value to true inside the method. The other will return raw value without having conversion. The overview of these methods is as following table, and will describe in the following section.

Name of Methods
AO_ GetSafetyValue (DWORD* o_ dwSafetyValue)
AO_ GetSafetyValue (float* o_ fSafetyValue)

5.6.6.11.1 AO_ GetSafetyValue (DWORD*)

Analog output function - Get Safety value.Each channel takes one unit of DWORD array.

Syntax

```
public int AO_ GetSafetyValue (
    DWORD* o_dwSafetyValue
)
```

Parameters

*o_ dwSafetyValue

[OUT] The DWORD array of Safety Value

Return Value

Error code

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
DWORD o_dwSafetyValue [USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{
if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetSafetyValue (o_dwSafetyValue)))
    printf("%d", iErrCode);
else
{
for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
      printf("0x%04x\n", o_dwSafetyValue [iIdx]);
}
}
return iErrCode;
```

5.6.6.11.2 AO_ GetSafetyValue (float*)

Analog output function - Get Safety value.Each channel takes one unit of float array.

Syntax

```
public int AO_ GetSafetyValue (
    float* o_ fSafetyValue
)
```

Parameters

*o_ fSafetyValue

[OUT] The float array of Safety Value

Return Value

Error code

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
float o_fSafetyValue [USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{
if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetSafetyValue (o_ fSafetyValue)))
    printf("%d", iErrCode);
else
{
for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
      printf("%04f\n", o_fSafetyValue [iIdx]);
}
}
return iErrCode;
```

5.6.6.12 AO_SetTypeCode

The class has two overload methods for setting type code. One provides specifying channel to set, another for all channel. Please refer to user's manual for analog output type code. These two overload methods are listed as following table and described in following section. The corresponding type code can be found in Appendix A.2.

Name of Methods
AO_SetTypeCode (BYTE i_byChToSet, BYTE i_byTypeCode)
AO_SetTypeCode (BYTE *i_byTypeCodes)

5.6.6.12.1 AO_SetTypeCode (BYTE, BYTE)

Analog output function - Set type code for specific channel. The type code can reference to Appendix A.2.

Syntax

```
public int AO_SetTypeCode (
    BYTE i_byChToSet,
    BYTE i_byTypeCode
)
```

Parameters

i_byChToSet

[IN] The specific channel to set

i_byTypeCode

[IN] The type code for the specific channel

Return Value

Error code

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
BYTE o_byTypeCode [USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_SetTypeCode(0, 0x30)))
      printf(  "%d"  , iErrCode);
   iErrCode = m_usbIO.CloseDevice();
}
else
return iErrCode;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. AO_GetTypeCode (o_byTypeCode)))
     printf("%d", iErrCode);
   else
   {
      for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
      printf("0x%02x\n", o_byTypeCode[iIdx]);
   }
}
return iErrCode;
```

5.6.6.12.2 AO_SetTypeCode (BYTE*)

Analog output function - Set type code for all channels. The type code can reference to Appendix A.2.

Syntax

```
public int AO_SetTypeCode (
    BYTE *i_byTypeCodes
)
```

Parameters

*i_byTypeCodes

[IN] The byte array of type code to set

Return Value

Error code

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
BYTE o_byTypeCode [USBIO_AO_MAX_CHANNEL];
BYTE m_byChTypeCode[USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{

   for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
    {
      m_byChTypeCode[iIdx] = 0x30;
    }

   if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_SetTypeCode(m_byChTypeCode)))
      printf("%d", iErrCode);

   if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetTypeCode(o_byTypeCode)))
       printf("%d", iErrCode);
   else
   {
    for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
         printf("0x%02x\n", o_byTypeCode[iIdx]);
   }

   iErrCode = m_usbIO.CloseDevice();
}
return iErrCode;
```

5.6.6.13 AO_SetChEnable

Analog output function - Set channel enable/disable. Each byte indicates 8 channels enable/disable mask. Ex: Byte0 -> Channel0 ~ 7

Syntax

```
public int AO_SetChEnable (
    BYTE *i_byChEnable
)
```

Parameters

*i_byChEnable

[IN] The byte array of channel enable/disable mask

Return Value

Error code

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
BYTE m_byChEnable[(USBIO_AO_MAX_CHANNEL + 7) / 8];
BYTE o_byChEnable[(USBIO_AO_MAX_CHANNEL + 7) / 8];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{
   for(iIdx = 0; iIdx <( USBIO_AO_MAX_CHANNEL + 7) / 8; iIdx++)
      m_byChEnable [iIdx] = 0x03;
if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_SetChEnable(m_byChEnable)))
   printf("%d", iErrCode);

if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetChEnable (o_byChEnable)))
    printf("%d", iErrCode);
else
{
  for(iIdx = 0; iIdx < (USBIO_AO_MAX_CHANNEL + 7) / 8; iIdx++)
      printf("0x%02x\n", o_byChEnable [iIdx]);
}

iErrCode = m_usbIO.CloseDevice();
}
return iErrCode;
```

5.6.6.14 AO_ WriteValue

The class library provides 4 overload methods to write AO expect value. Two method provides specifying channel to write raw data and converted value.The others write AO value for all channels.The overview of these methods is as following table, and will describe in the following section.

Name of Methods
AO_ WriteValue (BYTE i_byChToSet, DWORD i_dwAOVal)
AO_ WriteValue (DWORD* i_dwAOValue)
AO_ WriteValue (BYTE i_byChToSet, float i_fAOExpValue)
AO_ WriteValue (float* i_fAOExpValue)

5.6.6.14.1 AO_WriteValue (BYTE, DWORD)

Analog output function - Write AO expect value to specifying channel in double word (digital) format.

In the digital format, the value represents the value from zero to full scale. Ex: For type -10V ~ +10V, the value 0x0 indicates -10V and 0xFFFF (16bit resolution) indicates +10V.

Please note that, when channel was not in good status, the reading value no longer represents zero to full scale. Different channel status follows the following rule:

- Channel Under
  The reading value represents a sign value X indicates how many value under zero scale range. This value can be calculated by following formula:
  Assume current type is -5V ~ +5V with 16 bit resolution and reading value is 0x53E, then we can get the actual value Y is $\mathrm{Y} = \left(0 - \frac{0\mathrm{X53E}}{0\mathrm{xFFFF}}\right) \times \left(5 - (-5)\right) + (-5)$
- Channel Open & Channel Close
  The reading value of these two statuses will be the full scale for channel open and zero scale for channel close.

Syntax

```
public int AO_WriteValue(
    BYTE i_byChToSet,
    DWORD i_dwAOVal
);
```

Parameters

i_byChToSet

[IN] The specific AO channel to be set.

i_dwAOVal

[IN] The AO value .

Return Value

Error code

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
DWORD o_dwAOExpValue[USBIO_AO_MAX_CHANNEL];
BYTE byChannel = 0;
DWORD dwAOValue = 0xff;
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{
if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_WriteValue(byChannel, dwAOValue)))
   printf("%d", iErrCode);

if(ERR_NO_ERR != (iErrCode = m_usbIO. AO_ReadExpValue(o_dwAOExpValue)))
   printf("%d", iErrCode);
else
{
   for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
   printf("0x%04x\n", o_dwAOExpValue[iIdx]);
}
}
return iErrCode;
```

5.6.6.14.2 AO_WriteValue (DWORD*)

Analog output function - Write AO expect value to all channels in double word (digital) format.

Syntax

```
public int AO_WriteValue(
    DWORD* i_dwAOVal
);
```

Parameters

i_dwAOVal

[IN] The AO value .

Return Value

Error code

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
DWORD o_dwAOExpValue[USBIO_AO_MAX_CHANNEL];
DWORD dwAOValue[USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{
for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
  dwAOValue[iIdx] = 0xff;

if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_WriteValue(dwAOValue)))
    printf("%d", iErrCode);

if(ERR_NO_ERR != (iErrCode = m_usbIO. AO_ReadExpValue(o_dwAOExpValue)))
    printf("%d", iErrCode);
else
{
    for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
    printf("0x%04x\n", o_dwAOExpValue[iIdx]);
}
}
return iErrCode;
```

5.6.6.14.3 AO_WriteValue (BYTE, float)

Analog output function - Write AO expect value to specifying channel in float (analog) format.

The writting value is calculated, users write real value for current output type. Ex: The writting value is 1.316 in -2.5 ~ +2.5V, the output signal is 1.316V.

Syntax

```
public int AO_WriteValue(
    BYTE i_byChToSet,
    DWORD i_fAOVal
);
```

Parameters

i_byChToSet

[IN] The specific AO channel to be set.

i_fAOVal

[IN] The AO value .

Return Value

Error code

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
float o_fAOExpValue[USBIO_AO_MAX_CHANNEL];
BYTE byChannel = 0;
float fAOValue = 5;
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{
if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_WriteValue(byChannel, fAOValue)))
   printf("%d", iErrCode);

if(ERR_NO_ERR != (iErrCode = m_usbIO. AO_ReadExpValue(o_fAOExpValue)))
   printf("%d", iErrCode);
else
{
   for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
   printf("%04f\n", o_fAOExpValue[iIdx]);
}
}
return iErrCode;
```

5.6.6.14.4 AO_WriteValue (float*)

Analog output function - Write AO expect value to all channels in float (analog) format.

The writting value is calculated, users write real value for current output type. Ex: The writting value is 1.316 in -2.5 ~ +2.5V, the output signal is 1.316V.

Syntax

```
public int AO_WriteValue(
    float* i_fAOVal
);
```

Parameters

i_ fAOVal

[IN] The AO value .

Return Value

Error code

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
float o_fAOExpValue[USBIO_AO_MAX_CHANNEL];
float fAOValue[USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{
for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
  fAOValue[iIdx] = 5;

if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_WriteValue(fAOValue)))
    printf("%d", iErrCode);

if(ERR_NO_ERR != (iErrCode = m_usbIO. AO_ReadExpValue(o_fAOExpValue)))
    printf("%d", iErrCode);
else
{
    for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
    printf("%04f\n", o_fAOExpValue[iIdx]);
}
}
return iErrCode;
```

5.6.6.15 AO_SetPowerOnEnable

Analog output function - Set Power-On Enable. Each channel takes one byte.

Syntax

```
public int AO_SetPowerOnEnable (
    BYTE *o_byPowerOnEnable
)
```

Parameters

*o_byPowerOnEnable

[OUT] The byte array of channel enable/disable mask

Return Value

Error code

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
BYTE o_byPwrOnEnable [USBIO_AO_MAX_CHANNEL];
BYTE i_bySetPwrOnEnable[USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{
  for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
    i_bySetPwrOnEnable[iIdx] = 0x01;

  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_SetPowerOnEnable(i_bySetPwrOnEnable)))
     printf("%d", iErrCode);

if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetPowerOnEnable (o_byPwrOnEnable)))
     printf("%d", iErrCode);
else
{
for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
       printf("0x%02x\n", o_byPwrOnEnable [iIdx]);
}
}
return iErrCode;
```

5.6.6.16 AO_SetSafetyEnable

Analog output function - Set Safety Enable. Each byte indicates 8 channels enable/disable mask. EX: Byte0 -> Channel0 ~ 7.

Syntax

```
public int AO_SetSafetyEnable (
    BYTE *o_bySafetyEnable
)
```

Parameters

*o_bySafetyEnable

[OUT] The byte array of channel enable/disable mask

Return Value

Error code

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
BYTE o_bySafetyEnable [(USBIO_AO_MAX_CHANNEL + 7) / 8];
BYTE i_bySetSafetyEnable[(USBIO_AO_MAX_CHANNEL + 7) / 8] = {0x03};
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_SetSafetyEnable(i_bySetSafetyEnable)))
     printf("%d", iErrCode);

  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetSafetyEnable (o_bySafetyEnable)))
       printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx < (USBIO_AO_MAX_CHANNEL + 7) / 8; iIdx++)
        printf("0x%02x\n", o_bySafetyEnable [iIdx]);
  }
}
return iErrCode;
```

5.6.6.17 AO_ SetPowerOnValue

The class library provides 4 overload methods to write AO Power-On Value. Two method provides specifying channel to write raw data and converted value.The others write AO Power-On Value for all channels.The overview of these methods is as following table, and will describe in the following section.

Name of Methods
AO_ SetPowerOnValue (BYTE i_byChToSet, DWORD i_dwPowerOnValue)
AO_ SetPowerOnValue (DWORD* i_dwPowerOnValue)
AO_ SetPowerOnValue (BYTE i_byChToSet, float i_fPowerOnValue)
AO_ SetPowerOnValue (float* i_fPowerOnValue)

5.6.6.17.1 AO_ SetPowerOnValue (BYTE, DWORD)

Analog output function - Set AO Power-On Value to specifying channel in double word (digital) format.

In the digital format, the value represents the value from zero to full scale. Ex: For type -10V ~ +10V, the value 0x0 indicates -10V and 0xFFFF (16bit resolution) indicates +10V.

Please note that, when channel was not in good status, the reading value no longer represents zero to full scale. Different channel status follows the following rule:

- Channel Under
  The reading value represents a sign value X indicates how many value under zero scale range. This value can be calculated by following formula:
  Assume current type is -5V ~ +5V with 16 bit resolution and reading value is 0x53E, then we can get the actual value Y is $\mathrm{Y} = \left(0 - \frac{0\mathrm{X53E}}{0\mathrm{xFFFF}}\right) \times \left(5 - (-5)\right) + (-5)$
- Channel Open & Channel Close
  The reading value of these two statuses will be the full scale for channel open and zero scale for channel close.

Syntax

```
public int AO_ SetPowerOnValue (
    BYTE i_byChToSet,
    DWORD i_dwPowerOnValue
);
```

Parameters

i_byChToSet

[IN] The specific AO channel to be set.

i_dwPowerOnValue

[IN] The AO PowerOnValue .

Return Value

Error code

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
DWORD o_dwPowerOnValue [USBIO_AO_MAX_CHANNEL];
BYTE byChannel = 0;
DWORD dwSetPowerOnValue = 0xff;
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{
if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_SetPowerOnValue (byChannel, dwSetPowerOnValue)))
   printf("%d", iErrCode);

if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetPowerOnValue (o_dwPowerOnValue)))
   printf("%d", iErrCode);
else
{
   for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
   printf("0x%04x\n", o_dwPowerOnValue[iIdx]);
}
}
return iErrCode;
```

5.6.6.17.2 AO_ SetPowerOnValue (DWORD*)

Analog output function - Set AO Power-On Value to all channels in double word (digital) format.

Syntax

```
public int AO_ SetPowerOnValue (
    DWORD* i_ dwPowerOnValue
);
```

Parameters

i_ dwPowerOnValue

[IN] The AO Power-On Value.

Return Value

Error code

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
DWORD o_dwPowerOnValue [USBIO_AO_MAX_CHANNEL];
DWORD dwSetPowerOnValue[USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{
for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
   dwSetPowerOnValue[iIdx] = 0xff;

if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_SetPowerOnValue (dwSetPowerOnValue)))
   printf("%d", iErrCode);

if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetPowerOnValue (o_dwPowerOnValue)))
   printf("%d", iErrCode);
else
{
   for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
   printf("0x%04x\n", o_dwPowerOnValue[iIdx]);
}
}
return iErrCode;
```

5.6.6.17.3 AO_ SetPowerOnValue (BYTE, float)

Analog output function - Set AO Power-On Value to specifying channel in float (analog) format.

The writting value is calculated, users write real value for current output type. Ex: The writting value is 1.316 in -2.5 ~ +2.5V, the output signal is 1.316V.

Syntax

```
public int AO_ SetPowerOnValue (
    BYTE i_byChToSet,
    DWORD i_fPowerOnValue
);
```

Parameters

i_byChToSet

[IN] The specific AO channel to be set.

i_ fPowerOnValue

[IN] The AO Power-On Value.

Return Value

Error code

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
float o_fPowerOnValue [USBIO_AO_MAX_CHANNEL];
BYTE byChannel = 0;
float fSetPowerOnValue = 5;
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{
if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_SetPowerOnValue (byChannel, fSetPowerOnValue)))
   printf("%d", iErrCode);

if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetPowerOnValue (o_fPowerOnValue)))
   printf("%d", iErrCode);
else
{
   for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
   printf("%04f\n", o_fPowerOnValue[iIdx]);
}
}
return iErrCode;
```

5.6.6.17.4 AO_ SetPowerOnValue (float*)

Analog output function - Set AO Power-On Value to all channels in float (analog) format.

The writting value is calculated, users write real value for current output type. Ex: The writting value is 1.316 in -2.5 ~ +2.5V, the output signal is 1.316V.

Syntax

```
public int AO_ SetPowerOnValue (
    float* i_ fPowerOnValue
);
```

Parameters

i_ fPowerOnValue

[IN] The AO Power-On Value.

Return Value

Error code

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
float o_fPowerOnValue [USBIO_AO_MAX_CHANNEL];
float fSetPowerOnValue[USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{
for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
fSetPowerOnValue[iIdx] = 5;

if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_SetPowerOnValue (fSetPowerOnValue)))
   printf("%d", iErrCode);

if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetPowerOnValue (o_fPowerOnValue)))
   printf("%d", iErrCode);
else
{
   for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
   printf("%04f\n", o_fPowerOnValue[iIdx]);
}
}
return iErrCode;
```

5.6.6.18 AO_ SetSafetyValue

The class library provides 4 overload methods to write AO Safety Value. Two method provides specifying channel to write raw data and converted value.The others write AO Safety Value for all channels.The overview of these methods is as following table, and will describe in the following section.

Name of Methods
AO_ SetSafetyValue (BYTE i_byChToSet, DWORD i_dwSafetyValue)
AO_ SetSafetyValue (DWORD* i_dwSafetyValue)
AO_ SetSafetyValue (BYTE i_byChToSet, float i_fSafetyValue)
AO_ SetSafetyValue (float* i_fSafetyValue)

5.6.6.18.1 AO_ SetSafetyValue (BYTE, DWORD)

Analog output function - Set AO Safety Value to specifying channel in double word (digital) format.

In the digital format, the value represents the value from zero to full scale. Ex: For type -10V ~ +10V, the value 0x0 indicates -10V and 0xFFFF (16bit resolution) indicates +10V.

Please note that, when channel was not in good status, the reading value no longer represents zero to full scale. Different channel status follows the following rule:

- Channel Under
  The reading value represents a sign value X indicates how many value under zero scale range. This value can be calculated by following formula:
  Assume current type is -5V ~ +5V with 16 bit resolution and reading value is 0x53E, then we can get the actual value Y is $\mathrm{Y} = \left(0 - \frac{0\mathrm{X53E}}{0\mathrm{xFFFF}}\right) \times \left(5 - (-5)\right) + (-5)$
- Channel Open & Channel Close
  The reading value of these two statuses will be the full scale for channel open and zero scale for channel close.

Syntax

```
public int AO_ SetSafetyValue (
    BYTE i_byChToSet,
    DWORD i_dwSafetyValue
);
```

Parameters

i_byChToSet

[IN] The specific AO channel to be set.

i_dwSafetyValue

[IN] The AO Safety Value.

Return Value

Error code

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
DWORD o_dwSafetyValue [USBIO_AO_MAX_CHANNEL];
BYTE byChannel = 0;
DWORD dwSetSafetyValue = 0xff;
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{
if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_SetPowerOnValue (byChannel, dwSetSafetyValue)))
   printf("%d", iErrCode);

if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetPowerOnValue (o_dwSafetyValue)))
   printf("%d", iErrCode);
else
{
   for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
   printf("0x%04x\n", o_dwSafetyValue[iIdx]);
}
}
return iErrCode;
```

5.6.6.18.2 AO_ SetSafetyValue (DWORD*)

Analog output function - Set AO Safety Value to all channels in double word (digital) format.

Syntax

```
public int AO_ SetSafetyValue (
    DWORD* i_ dwSafetyValue
);
```

Parameters

i_ dwSafetyValue

[IN] The AO Safety Value.

Return Value

Error code

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
DWORD o_dwSafetyValue [USBIO_AO_MAX_CHANNEL];
DWORD dwSetSafetyValue [USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{
for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
dwSetSafetyValue[iIdx] = 0xff;

if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_SetPowerOnValue (dwSetSafetyValue)))
   printf("%d", iErrCode);

if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetPowerOnValue (o_dwSafetyValue)))
   printf("%d", iErrCode);
else
{
   for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
   printf("0x%04x\n", o_dwSafetyValue[iIdx]);
}
}
return iErrCode;
```

5.6.6.18.3 AO_ SetSafetyValue (BYTE, float)

Analog output function - Set AO Safety Value to specifying channel in float (analog) format.

The writting value is calculated, users write real value for current output type. Ex: The writting value is 1.316 in -2.5 ~ +2.5V, the output signal is 1.316V.

Syntax

```
public int AO_ SetSafetyValue (
    BYTE i_byChToSet,
    DWORD i_fSafetyValue
);
```

Parameters

i_byChToSet

[IN] The specific AO channel to be set.

i_ fSafetyValue

[IN] The AO Safety Value.

Return Value

Error code

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
float o_fSafetyValue [USBIO_AO_MAX_CHANNEL];
BYTE byChannel = 0;
float fSetSafetyValue = 5;
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{
if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_SetPowerOnValue (byChannel, fSetSafetyValue)))
   printf("%d", iErrCode);

if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetPowerOnValue (o_fSafetyValue)))
   printf("%d", iErrCode);
else
{
   for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
   printf("%04f\n", o_fSafetyValue[iIdx]);
}
}
return iErrCode;
```

5.6.6.18.4 AO_ SetSafetyValue (float*)

Analog output function - Set AO Safety Value to all channels in float (analog) format.

The writting value is calculated, users write real value for current output type. Ex: The writting value is 1.316 in -2.5 ~ +2.5V, the output signal is 1.316V.

Syntax

```
public int AO_ SetSafetyValue (
    float* i_ fSafetyValue
);
```

Parameters

i_ fSafetyValue

[IN] The AO Safety Value.

Return Value

Error code

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
float o_fSafetyValue [USBIO_AO_MAX_CHANNEL];
float fSetSafetyValue [USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2026, 1)))
{
for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
  fSetSafetyValue[iIdx] = 5;

if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_SetPowerOnValue (fSetSafetyValue)))
    printf("%d", iErrCode);

if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetPowerOnValue (o_fSafetyValue)))
    printf("%d", iErrCode);
else
{
    for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
    printf("%04f\n", o_fSafetyValue[iIdx]);
}
}
return iErrCode;
```

5.6.7 Pulse Input

5.6.7.1 PI_GetTotalSupportType

Pulse input function - Get total supported amount.

Syntax

```
public int PI_GetTotalSupportType (
    BYTE * o_byTotalSupportType
)
```

Parameters

* o_byTotalSupportType

  [OUT] The number of total support type

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byTotalSupportType;
Byte o_bySupportTypeCode[USBIO_MAX_SUPPORT_TYPE];
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2084, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. PI_GetTotalSupportType (&o_ byTotalSupportType)))
   {
      printf( "%d" , iErrCode);
      bRet = false;
   }
   if(ERR_NO_ERR != (iErrCode = m_usbIO. PI_GetSupportTypeCode (o_ bySupportTypeCode)))
   {
      printf( "%d" , iErrCode);
      bRet = false;
   }
   If(bRet)
   {
      printf( "%d\n" , o_byTotalSupportType);
      for(iIdx = 0; iIdx < o_byTotalSupportType; iIdx++)
         printf( "%02x\n" , o_bySupportTypeCode[iIdx]);
   }
}
```

5.6.7.2 PI_GetSupportTypeCode

Pulse input function - Get supported type code. Please refer to Appendix A.3 of user's manual to map PI channels input type.

Syntax

```
public int PI_GetTotalSupportType (
    BYTE *o_bySupportTypeCode
)
```

Parameters

*o_byTotalSupportType

[OUT] The number of total support type

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byTotalSupportType;
Byte o_bySupportTypeCode[USBIO_MAX_SUPPORT_TYPE];
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2084, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. PI_GetTotalSupportType (&o_ byTotalSupportType)))
   {
      printf( "%d" , iErrCode);
      bRet = false;
   }
   if(ERR_NO_ERR != (iErrCode = m_usbIO. PI_GetSupportTypeCode (o_ bySupportTypeCode)))
   {
      printf( "%d" , iErrCode);
      bRet = false;
   }
   If(bRet)
   {
      printf( "%d\n" , o_byTotalSupportType);
      for(iIdx = 0; iIdx < o_byTotalSupportType; iIdx++)
         printf( "%02x\n" , o_bySupportTypeCode[iIdx]);
   }
}
```

5.6.7.3 PI_GetTypeCode

Pulse input function - Get type code. Please refer to user's manual to map PI channels input type. The type code can reference to Appendix A.3.

Syntax

```
public int PI_GetTypeCode (
    BYTE *o_byTypeCode
)
```

Parameters

*o_byTypeCode

[OUT] The byte array of type code

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byTypeCode [USBIO_PI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2084, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. PI_GetTypeCode (o_byTypeCode)))
      printf( "%d" , iErrCode);
   else
   {
      for(iIdx = 0; iIdx < USBIO_PI_MAX_CHANNEL; iIdx++)
         printf( "%02x\n" , o_byTypeCode[iIdx]);
   }
}
```

5.6.7.4 PI_GetTriggerMode

Pulse input function - Get trigger mode

Trigger Mode	Code
Falling edge	0
Rising edge	1
Both edge	2 & 3

Syntax

```
public int PI_GetTriggerMode (
    BYTE *o_byTriggerMode
)
```

Parameters

*o_byTriggerMode

[OUT] The byte array of trigger mode

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byTriggerMode [USBIO_PI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2084, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. PI_GetTriggerMode (o_byTriggerMode)))
      printf( "%d" , iErrCode);
   else
   {
      for(iIdx = 0; iIdx < USBIO_PI_MAX_CHANNEL; iIdx++)
         printf( "%02x\n" , o_byTriggerMode [iIdx]);
   }
}
```

5.6.7.5 PI_GetChIsolatedFlag

Pulse input function - Get channel isolated flag. Each byte indicates 8 channels flag. EX: Byte0 -> Channel0 ~ 7.

Syntax

```
public int PI_GetChIsolatedFlag (
    BYTE *o_byChIsolatedFlag
)
```

Parameters

*o_byChIsolatedFlag

[OUT] The byte arrays of channel isolated flag

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byChIsolatedFlag [(USBIO_PI_MAX_CHANNEL + 7) / 8];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2084, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.PI_GetChIsolatedFlag(o_byChIsolatedFlag)))
      printf( "%d" , iErrCode);
   else
   {
      for(iIdx = 0; iIdx < (USBIO_PI_MAX_CHANNEL + 7) / 8; iIdx++)
         printf( "%02x\n" , o_byChIsolatedFlag[iIdx]);
   }
}
```

5.6.7.6 PI_GetLPFilterEnable

Pulse input function - Get low-pass filter enable. Each byte indicates 8 channels enable/disable mask. EX: Byte0 -> Channel0 ~ 7.

Syntax

```
public int PI_GetLPFilterEnable (
    BYTE * o_byLPFilterEnable
)
```

Parameters

* o_byLPFilterEnable

    [OUT] The byte array of the low-pass filter enable mask

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byLPFilterEnable [(USBIO_PI_MAX_CHANNEL + 7) / 8];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2084, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.PI_GetLPFilterEnable(o_byLPFilterEnable)))
      printf( "%d" , iErrCode);
   else
   {
      for(iIdx = 0; iIdx < (USBIO_PI_MAX_CHANNEL + 7) / 8; iIdx++)
         printf( "%02x\n" , o_byLPFilterEnable[iIdx]);
   }
}
```

5.6.7.7 PI_GetLPFilterWidth

Pulse input function - Get low-pass filter width. The unit of the width is uS. The maximum value of width is 32767uS.

Note: Each channel does not use own low-pass filter width. Please refer to following table to see what low-pass filter width is referred to.

Channel Index	Set
0 & 1	0
2 & 3	1
4, 5, 6, 7	2

Syntax

```
public int PI_GetLPFilterWidth (
    WORD *o_wLPFilterWidth
)
```

Parameters

*o_wLPFilterWidth

[OUT] The byte array of the low-pass filter width in uS

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
WORD o_wLPFilterWidth [USBIO_PI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2084, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.PI_GetLPFilterWidth(o_wLPFilterWidth)))
      printf( "%d" , iErrCode);
   else
   {
      for(iIdx = 0; iIdx < USBIO_PI_MAX_CHANNEL; iIdx++)
         printf( "%d\n" , o_wLPFilterWidth[iIdx]);
   }
}
```

5.6.7.8 PI_ReadValue

Pulse input function - Get PI value in double-word format. This method provides two formats in a function call.

NOTE: If the type of the channel is frequency, users have to convert these 4 bytes into float format.

Syntax

```
public int PI_ReadValue (
    DWORD *o_dwPIValue
    BYTE *o_byChStatus
)
```

Parameters

*o_dwPIValue

[OUT] The byte array of the PI channel counter value

*o_byChStatus

[OUT] The byte array of the channel status

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
DWORD o_dwPIValue[USBIO_PI_MAX_CHANNEL];
BYTE o_byChStatus[USBIO_PI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2084, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. PI_ReadValue(o_dwPIValue, o_byChStatus)))
      printf( "%d" , iErrCode);
   else
   {
      for(iIdx = 0; iIdx < USBIO_PI_MAX_CHANNEL; iIdx++)
         printf( "%d\n" , o_dwPIValue[iIdx]);
   }
}
```

5.6.7.9 PI_ReadCntValue

Pulse input function - Read PI value in double-word format. This method reads the all counter value of channels.

NOTE: If the channel is in the type of frequency. The value of the related channel of the o_dwCntValue will be 0, and the value of the related channels of o_byChStatus will indicate the type not support.

Syntax

```
public int PI_ReadValue (
    DWORD *o_dwCntValue
    BYTE *o_byChStatus
)
```

Parameters

*o_dwCntValue

[OUT] The unsigned long array of the PI channel counter value

*o_byChStatus

[OUT] The byte array of the channel status

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
DWORD o_dwCntValue[USBIO_PI_MAX_CHANNEL];
BYTE o_byChStatus[USBIO_PI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2084, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. PI_ReadCntValue(o_dwCntValue, o_byChStatus)))
      printf( "%d" , iErrCode);
   else
   {
      for(iIdx = 0; iIdx < USBIO_PI_MAX_CHANNEL; iIdx++)
         printf( "%d\n" , o_dwCntValue[iIdx]);
   }
}
```

5.6.7.10 PI_ReadFreqValue

Pulse input function - Read the frequency value. This method reads the all frequency value of channels.

NOTE: If the channel is not in the type of frequency. The value of the related channel of the o_dwCntValue will be -1, and the value of the related channels of o_byChStatus will indicate the type not support.

Syntax

```
public int PI_ReadValue (
    float *o_fFreqValue
    BYTE *o_byChStatus
)
```

Parameters

*o_fFreqValue

[OUT] The float array of the PI channel frequency value

*o_byChStatus

[OUT] The byte array of the channel status

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
float o_fFreqValue[USBIO_PI_MAX_CHANNEL];
BYTE o_byChStatus[USBIO_PI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2084, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO. PI_ReadValue(o_fFreqValue, o_byChStatus)))
      printf( "%d" , iErrCode);
   else
   {
      for(iIdx = 0; iIdx < USBIO_PI_MAX_CHANNEL; iIdx++)
         printf( "%f\n" , o_fFreqValue[iIdx]);
   }
}
```

5.6.7.11 PI_ReadBulkValue

Analog input function – Trigger reading bulk PI value (Fast acquire functionality).

When in callback operation, it will cause the performance in your callback function. Please reduce execute time in this callback function.

The detail of operation is described as follow. When call this API, the PI module operation will be changed from normal to fast acquire mode. In fast acquire mode, PI module follow the parameter of API setting to acquire data.

The API has block and non-block operation. In block operation, user’ s application needs to wait until API finishing all procedure. In contrast with block mode, non-block provides a flexible way for user. In non-block operation, user’ s application can proceed to own other code. To enable non-block operation, it is important to declare a callback function and pass it through last parameter. For block operation, just pass a NULL definition in last parameter.

Due to the USB 2.0 Full-speed transfer rate capability, the maximum sample rate is 10 KHz.

Syntax

```
public int PI_ReadBulkValue (
    BYTE i_byStartCh,
    BYTE i_byChTotal,
    DWORD i_dwSampleWidth,
    Float i_fSampleRate,
    DWORD i_dwBufferWidth,
    DWORD *o_dwDataBuffer,
    OnBulkValueFinishEvent i_CBFunc
)
```

Parameters

i_byStartCh

[IN] The starting acquire channel

i_byChTotal

[IN] The total channels to acquire

i_dwSampleWidth

[IN] The sampling width (ms)

i_fSampleRate

[IN] The sampling rate (Hz). 10KHz maximum.

i_dwBufferWidth

[IN] The width of the buffer for single channel

*o_dwDataBuffer

[OUT] The 2-dimension buffer array to store

i_CBFunc

[IN] Block operation – NULL

[IN] Non-block operation - The address of callback function.

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
// To read 0~1 channel for 100ms in 5 KHz sample rate each channel in non-block operation
// So we have the following variable declaration
#define SampleRate 5000
#define BufferWidth 500; // 5000(Hz) * 0.1(100ms)
DWORD m_dwBuffer[2][BufferWidth];

Void BulkFinishCallback(DWORD dwCount)
{
   // Callback function to handle data
}

Int main()
{
   m_usbIO = new ICPDAS_USBIO();
   if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2084, 1)))
   {
      if(ERR_NO_ERR != (iErrCode = m_usbIO.PI_ReadBulkValue(0,
                                                            2,
                                                            100,
                                                            SampleRate,
                                                            BufferWidth
                                                            m_dwBuffer,
                                                            BulkFinishCallback)))
         printf( "%d" , iErrCode);

      while(1) {Sleep(1);}
   }
}
```

5.6.7.12 PI_SetTypeCode

The class has two overload methods for setting type code. One provides specifying channel to set, another for all channels. Please refer to user's manual for pulse input type code. These two overload methods are listed as following table and described in following section. The corresponding type code can be found in Appendix A.3.

Name of Methods
PI_SetTypeCode (BYTE i_byChToSet, BYTE i_byTypeCode)
PI_SetTypeCode (BYTE *i_byTypeCodes)

5.6.7.12.1 PI_SetTypeCode (BYTE, BYTE)

Pulse input function - Set type code for specific channel. The type code can reference to Appendix A.3.

Syntax

```
public int PI_SetTypeCode (
    BYTE i_byChToSet,
    BYTE i_byTypeCode
)
```

Parameters

i_byChToSet

[IN] The specific channel to set

i_byTypeCode

[IN] The type code for the specific channel

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;


m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2084, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.PI_SetTypeCode(0, 0x10)))
      printf( "%d" , iErrCode);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.7.12.2 PI_SetTypeCode (BYTE*)

Analog input function - Set type code for all channels. The type code can reference to Appendix A.3.

Syntax

```
public int PI_SetTypeCode (
    BYTE *i_byTypeCodes
)
```

Parameters

*i_byTypeCodes

[IN] The byte array of type code to set

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte m_byChTypeCode[USBIO_PI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2084, 1)))
{
   For(iIdx = 0; iIdx < USBIO_PI_MAX_CHANNEL; iIdx)
      m_byChTypeCode[iIdx] = 0x50;
   if(ERR_NO_ERR != (iErrCode = m_usbIO.PI_SetTypeCode(m_byChTypeCode)))
      printf( "%d" , iErrCode);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.7.13 PI_ClearChCount

Pulse input function - Clear channel count with clear mask. Each byte indicates 8 channels clear mask, set for channel clear. Ex: Byte0 -> Channel0 ~ 7

Syntax

```
public int PI_ClearChCount (
    BYTE *i_byClrMask
)
```

Parameters

*i_byClrMask

[IN] The byte array of channel count clear mask

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte m_byChClrMask[(USBIO_PI_MAX_CHANNEL + 7) / 8];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2084, 1)))
{
   For(iIdx = 0; iIdx <( USBIO_PI_MAX_CHANNEL + 7) / 8; iIdx)
      m_byChClrMask[iIdx] = 0x5A;
   if(ERR_NO_ERR != (iErrCode = m_usbIO.PI_ClearChCount(m_byChClrMask)))
      printf(  "%d" , iErrCode);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.7.14 PI_ClearSingleChCount

Pulse input function - Clear specific channel count.

Syntax

```
public int PI_ClearSingleChCount (
    BYTE i_byChToClr
)
```

Parameters

i_byChToClr

[IN] The channel index for clearing

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2084, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.PI_ClearSingleChCount(7)))
      printf( "%d" , iErrCode);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.7.15 PI_ClearChStatus

Pulse input function - Clear channel status with clear mask. Each byte indicates 8 channels clear mask, set for channel clear. Ex: Byte0 -> Channel0 ~ 7

Syntax

```
public int PI_ClearChStatus(
    BYTE *i_byClrMask
)
```

Parameters

*i_byClrMask

[IN] The byte array of channel status clear mask

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte m_byChClrMask[(USBIO_PI_MAX_CHANNEL + 7) / 8];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2084, 1)))
{
   For(iIdx = 0; iIdx <( USBIO_PI_MAX_CHANNEL + 7) / 8; iIdx)
      m_byChClrMask[iIdx] = 0x5A;
   if(ERR_NO_ERR != (iErrCode = m_usbIO.PI_ClearChStatus(m_byChClrMask)))
      printf(  "%d" , iErrCode);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.7.16 PI_ClearSingleChStatus

Pulse input function - Clear specific channel status.

Syntax

```
public int PI_ClearSingleChStatus(
    BYTE i_byChToClr
)
```

Parameters

i_byChToClr

[IN] The channel index for clearing

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2084, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.PI_ClearSingleChStatus(7)))
      printf( "%d" , iErrCode);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.7.17 PI_SetTriggerMode

The class has two overload methods for setting trigger mode. One provides specifying channel to set, another for all channels.

The trigger mode is shown as following table.

Trigger Mode	Code
Falling edge	0
Rising edge	1
Both edge	2 & 3

These two overload methods are listed as following table and described in following section.

Name of Methods
PI_SetTriggerMode (BYTE i_byChToSet, BYTE i_byTypeCode)
PI_SetTriggerMode (BYTE *i_byTypeCodes)

5.6.7.17.1 PI_SetTriggerMode (BYTE, BYTE)

Pulse input function - Set trigger mode to specific channel.

Syntax

```
public int PI_SetTriggerMode (
    BYTE i_byChToSet,
    BYTE i_byTriggerMode
)
```

Parameters

i_byChToSet

[IN] The specific channel to set

i_byTriggerMode

[IN] The type code for the specific channel

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2084, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.PI_SetTriggerMode(0, 0x1)))
      printf( "%d" , iErrCode);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.7.17.2 PI_SetTriggerMode (BYTE*)

Pulse input function - Set trigger mode to all channel.

Syntax

```
public int PI_SetTriggerMode (
    BYTE *i_byTriggerMode
)
```

Parameters

*i_byTriggerMode

[IN] The byte array of trigger mode to set

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte m_byChTriggerMode[USBIO_PI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2084, 1)))
{
   For(iIdx = 0; iIdx < USBIO_PI_MAX_CHANNEL; iIdx)
      m_byChTriggerMode[iIdx] = 0x2;
   if(ERR_NO_ERR != (iErrCode = m_usbIO.PI_SetTriggerMode(m_byChTriggerMode)))
      printf( "%d" , iErrCode);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.7.18 PI_SetChIsolatedFlag

The class has two overload methods for setting channel isolated flag. One provides specifying channel to set, the other for all channels. Set 1 for setting channel to isolated. The parameter of the method for all channels is constructed in byte array for all channel isolated flag, ex: Byte0 -> Channel0 ~ 7.

These two overload methods are listed as following table and described in following section.

Name of Methods
PI_SetChIsolatedFlag (BYTE i_byChToSet, BOOL i_bChIsolatedFlag)
PI_SetChIsolatedFlag (BYTE *i_byChIsolatedFlags)

5.6.7.18.1 PI_SetChIsolatedFlag (BYTE, BOOL)

Pulse input function - Set channel isolated flag.

Syntax

```
public int PI_SetChIsolatedFlag (
    BYTE i_byChToSet,
    BOOL i_bChIsolatedFlag
)
```

Parameters

i_byChToSet

[IN] The specific channel to set

i_bChIsolatedFlag

[IN] The isolated flag for the specific channel

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2084, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.PI_SetChIsolatedFlag(5, 0x1)))
      printf( "%d" , iErrCode);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.7.18.2 PI_SetChIsolatedFlag (BYTE*)

Pulse input function - Set channel isolated flag to all channels.

Syntax

```
public int PI_SetChIsolatedFlag (
    BYTE *i_byChIsolatedFlags
)
```

Parameters

*i_byChIsolatedFlags

[IN] The byte arrays of channel isolated flag.

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte m_byChIsolatedFlags[(USBIO_PI_MAX_CHANNEL + 7) / 8];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2084, 1)))
{
   For(iIdx = 0; iIdx < (USBIO_PI_MAX_CHANNEL + 7) / 8; iIdx)
      m_byChIsolatedFlag[iIdx] = 0x5a;
   if(ERR_NO_ERR != (iErrCode = m_usbIO.PI_SetChIsolatedFlag(m_byChIsolatedFlag)))
      printf( "%d" , iErrCode);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.7.19 PI_SetLPFilterEnable

The class has two overload methods for setting trigger mode. One provides specifying channel to set, another for all channels. Set 1 to enable low-pass filter. The parameter of the method for all channels is a byte array for all channel enable mask, ex: Byte0 -> Channel0 ~ 7.

These two overload methods are listed as following table and described in following section.

Name of Methods
PI_SetLPFilterEnable (BYTE i_byChToSet, BOOL i_bLPFilterEnable)
PI_SetLPFilterEnable (BYTE *i_byLPFilterEnable)

5.6.7.19.1 PI_SetLPFilterEnable (BYTE, BOOL)

Pulse input function - Set low-pass filter enable to specific channel.

Syntax

```
public int PI_SetLPFilterEnable (
    BYTE i_byChToSet,
    BOOL i_bLPFilterEnable
)
```

Parameters

i_byChToSet

[IN] The specific channel to set

i_bLPFilterEnable

[IN] The enable flag for the specific channel

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2084, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.PI_SetLPFilterEnable(5, 0x1)))
      printf( "%d" , iErrCode);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.7.19.2 PI_SetLPFilterEnable (BYTE*)

Pulse input function - Set low-pass filter enable to all channel.

Syntax

```
public int PI_SetLPFilterEnable (
    BYTE *i_byLPFilterEnable
)
```

Parameters

*i_byLPFilterEnable

[IN] The byte array of low-pass filter enable mask.

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte m_byLPFilterEnable[(USBIO_PI_MAX_CHANNEL + 7) / 8];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2084, 1)))
{
   For(iIdx = 0; iIdx < (USBIO_PI_MAX_CHANNEL + 7) / 8; iIdx)
      m_byLPFilterEnable[iIdx] = 0x5a;
   if(ERR_NO_ERR != (iErrCode = m_usbIO.PI_SetLPFilterEnable(m_byLPFilterEnable)))
      printf( "%d" , iErrCode);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.7.20 PI_SetLPFilterWidth

The class has two overload methods for setting trigger mode. One provides specifying channel to set, another for all channels. The low-pass filter width is for filtering noise or bouncing. The unit of the filter width is uS.

These two overload methods are listed as following table and described in following section.

Name of Methods
PI_SetLPFilterWidth (BYTE i_byChToSet, WORD i_wLPFilterWidth)
PI_SetLPFilterWidth (WORD *i_wLPFilterWidths)

5.6.7.20.1 PI_SetLPFilterEnable (BYTE, WORD)

Pulse input function - Set low-pass filter width

Syntax

```
public int PI_SetLPFilterWidth (
    BYTE i_byChToSet,
    BOOL i_wLPFilterWidth
)
```

Parameters

i_byChToSet

[IN] The specific channel to set

i_wLPFilterWidth

[IN] The low-pass filter width. (uS)

Return Value

Error code

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2084, 1)))
{
   if(ERR_NO_ERR != (iErrCode = m_usbIO.PI_SetLPFilterWidth(5, 10000)))
      printf( "%d" , iErrCode);
   iErrCode = m_usbIO.CloseDevice();
}
```

5.6.7.20.2 PI_SetLPFilterEnable (BYTE*)

Pulse input function - Set low-pass filter enable to all channel.

Syntax

```
public int PI_SetLPFilterEnable (
    BYTE *i_byLPFilterEnable
)
```

Parameters

*i_byLPFilterEnable

[IN] The byte array of low-pass filter enable mask.

Return Value

Error code

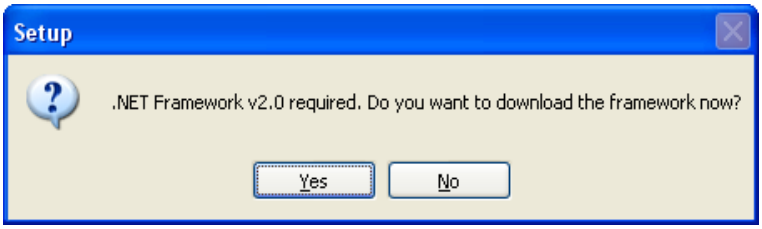
Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte m_byLPFilterWidth[USBIO_PI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB2084, 1)))
{
   For(iIdx = 0; iIdx < USBIO_PI_MAX_CHANNEL; iIdx)
      m_byLPFilterEnable[iIdx] = 20000;
   if(ERR_NO_ERR != (iErrCode = m_usbIO.PI_SetLPFilterWidth(m_byLPFilterWidth)))
      printf( "%d" , iErrCode);
   iErrCode = m_usbIO.CloseDevice();
}
```

6 Troubleshooting

1. **Cannot install ICP DAS USB I/O package with the message like the following figure.**



Because the ICP DAS USB I/O requires .NET Framework v2.0, the package will automatically detect the .NET Framework v2.0 installed as well or not. Users can click "Yes" to download and install the .NET Framework v2.0 via internet. If users can not access internet, the other way is install .NET Framework v2.0 in the folder "net_framework " in the root path of the CD.

2. **Returning timeout error code (65792) when access USB I/O.**

There are some possible reasons:

- The USB module connected to USB hub not local USB port on computer. This will cause the time for communication increased. To prevent this error, users can increase the time of communication timeout.
  (Note: We strongly recommended connecting USB modules to local USB port on computer to prevent unexpected problem.)
- Module is failure caused by unknown reason. You can refer to LED indicators section.

Appendix A

A.1 Analog Input Type Code

Code	Input Type	Code	Input Type
0x00	-15 mV ~ +15 mV	0x17	Type L TC, -200 ~ +800°C
0x01	-50 mV ~ + 50 mV	0x18	Type M TC, -200 ~ +100°C
0x02	-100 mV ~ +100 mV	0x19	Type $L_{DIN43710}$ TC, -200 ~ +900°C
0x03	-500 mV ~ +500 mV	0x1A	0 ~ +20 mA
0x04	-1 V ~ +1 V	0x1B	-150 V ~ +150 V
0x05	-2.5 V ~ +2.5 V	0x1C	-50 V ~ +50 V
0x06	-20 mA ~ +20 mA	0x20	Pt 100, α=.00385, -100 ~ +100°C
0x07	+4 mA ~ +20 mA	0x21	Pt 100, α=.00385, 0 ~ +100°C
0x08	-10 V ~ +10 V	0x22	Pt 100, α=.00385, 0 ~ +200°C
0x09	-5 V ~ +5 V	0x23	Pt 100, α=.00385, 0 ~ +600°C
0x0A	-1 V ~ +1 V	0x24	Pt 100, α=.003916, -100 ~ +100°C
0x0B	-500 mV ~ +500 mV	0x25	Pt 100, α=.003916, 0 ~ +100°C
0x0C	-150 mV ~ +150 mV	0x26	Pt 100, α=.003916, 0 ~ +200°C
0x0D	-20 mA ~ +20 mA	0x27	Pt 100, α=.003916, 0 ~ +600°C
0x0E	Type J TC, -210 ~ +760°C	0x28	Nickel 120, -80 ~ +100°C
0x0F	Type K TC, -210 ~ +1372°C	0x29	Nickel 120, 0 ~ +100°C
0x10	Type T TC, -270 ~ +400°C	0x2A	Pt 1000, α=.00392, -200 ~ +600°C
0x11	Type E TC, -270 ~ +1000°C	0x2B	Cu 100, α=.00421, -20 ~ +150°C
0x12	Type R TC, 0 ~ +1768°C	0x2C	Cu 100, α=.00427, 0 ~ +200°C
0x13	Type S TC, 0 ~ +1768°C	0x2D	Cu 1000, α=.00421, -20 ~ +150°C
0x14	Type B TC, 0 ~ +1820°C	0x2E	Pt 100, α=.00385, -200 ~ +200°C
0x15	Type N TC, -270 ~ +1300°C	0x2F	Pt 100, α=.003916, -200 ~ +200°C
0x16	Type C TC, 0 ~ +2320°C		

A.2 Analog Output Type Code

Code	Input Type
0x30	0 ~ +20 mA
0x31	4 ~ +20 mA
0x32	0 V ~ +10 V
0x33	-10 V ~ +10 V
0x34	0 V ~ +5 V
0x35	-5 V ~ +5 V

A.3 Pulse Input Type Code

Code	Input Type
0x50	Up counter
0x51	Frequency
0x52	Counter with battery backup
0x53	Encoder
0x54	Up/Down counter
0x55	Pulse/Direction counter
0x56	AB phase

A.4 Channel Status

Code	Input Type
0x00	Good
0x01	Over Range / Overflow
0x02	Under Range / Underflow
0x03	Open
0x04	Close
0x05	Type Not Supported

Appendix B

B.1 Error Codes

The error codes are divided into three parts, device, DEV-library and IO-library. Each part means different error code returned by device, DEV-library and IO-library. The error codes are described in the table.

Constant/Value	Description
ERR_NO_ERR 0x00000000, 0	The operation completed successfully.
ERR_DEV_ILLEGAL_FUNC 0x00000001, 1	The function is invalid.
ERR_ DEV_ILLEGAL_INDEX 0x00000002, 2	The index is invalid.
ERR_DEV_ILLEGAL_LENGTH 0x00000003, 3	The length is invalid.
ERR_DEV_ILLEGAL_PARAMETER 0x00000004, 4	The parameter is invalid.
ERR_DEV_ILLEGAL_MAPTABSIZE 0x00000005, 5	The size of mapping table is invalid.
ERR_DEV_ILLEGAL_MAPTABINDEX 0x00000006, 6	The index in mapping table is invalid.
ERR_ DEV_READONLY 0x00000007, 7	The index is read only.
ERR_ DEV_WRITEONLY 0x00000008, 8	The index is written only.
ERR_ DEV_BUFFERFULL 0x00000009, 9	The buffer in transceiver is full.
ERR_ DEV_LTTIMEOUT 0x0000000A, 10	The operation of large transfer has timeout.
ERR_ DEV_LTMODEFAIL 0x0000000B, 11	The current mode is not in large transfer mode.
ERR_ DEV_LTPKGLOST	The large transfer packet is lost.

0x0000000C, 12	
ERR_ DEV_LTINDEXNOTMACH 0x0000000D, 13	The offset index is not match while operating in large transfer.
ERR_ DEV_LTNOTFINISH 0x0000000E, 14	Another large transfer is operating.
ERR_DEV_DO_RELATED_ERR 0x00004000~0x000047FFF	The digital output related error in this region.
ERR_DEV_DI_RELATED_ERR 0x00004800~0x00004FFF	The digital input related error in this region.
ERR_DEV_AO_RELATED_ERR 0x00005000~0x000057FF	The analog output related error in this region.
ERR_DEV_AI_RELATED_ERR 0x00005800~0x00005FFF	The analog input related error in this region.
ERR_DEV_PO_RELATED_ERR 0x00006000~0x000067FF	The pulse output related error in this region.
ERR_DEV_PI_RELATED_ERR 0x00006800~0x00006FFF	The pulse input related error in this region.
ERR_USBDEV_INVALID_DEV 0x00010000, 65536	The handle of device is invalid.
ERR_USBDEV_DEV_OPENED 0x00010001, 65537	The device has been opened by class library.
ERR_USBDEV_DEVNOTEXISTS 0x00010002, 65538	The class library cannot find the device.
ERR_USBDEV_GETDEVINFO 0x00010003, 65539	An error was made to scan device.
ERR_USBDEV_ERROR_PKTSIZE 0x00010004, 65540	The packet size is invalid.
ERR_USBDEV_ERROR_WRITEFILE 0x00010004, 65541	An error occurs while writing packet to module.
ERR_USBIO_COMM_TIMEOUT 0x00010100, 65792	The communication between computer and device has been timeout.
ERR_USBIO_DEV_OPENED 0x00010101, 65793	The device has been opened by class library.
ERR_USBIO_DEV_NOTOPEN 0x00010102, 65794	The device has not opened for operating.
ERR_USBIO_INVALID_RESP 0x00010103, 65795	The data returned by device is invalid.

ERR_USBIO_IO_NOTSUPPORT 0x00010104, 65796	The method is not supported.
ERR_USBIO_PARA_ERROR 0x00010105, 65797	The parameter of method is invalid.
ERR_USBIO_BULKVALUE_ERR 0x00010106, 65798	An error occurs while getting bulk value.
ERR_USBIO_GETDEVINFO 0x00010107, 65799	An error occur while getting device information while device opening procedure.

Appendix C

C.1 Steps of updating firmware for USB I/O module

The USB I/O supports firmware updating by USB cable. Users can update firmware in their local site without sending module back. The following instruction describes how to update USB I/O firmware.

Step 1 Download the latest USB I/O installation package

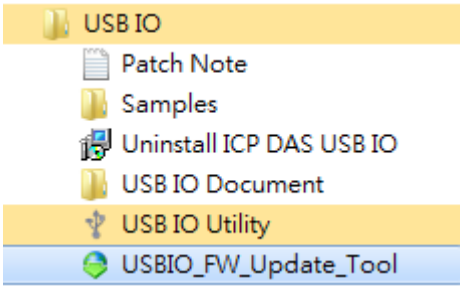
The latest installation package, version v1.0.0.8, adds an executable file for updating the USB I/O firmware. Users can find the file on the product page on the ICP DAS web site.

Step 2 Switch USB I/O module into INIT mode

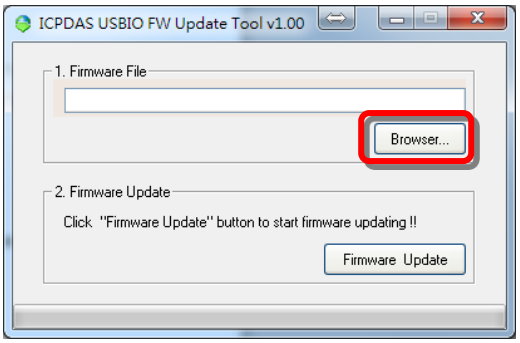
The USB I/O module only can update firmware when in INIT mode. Users have to switch module into INIT mode by the 2-way switch. Please refer to Section 2.1 Module Overview to have more information. After switching to INIT mode, uses MUST have to re-plug cable to reset USB I/O module.

Step 3 Execute the firmware update software and select firmware

The following figure is the USB I/O content in the start menu. Users can execute the firmware update software here.

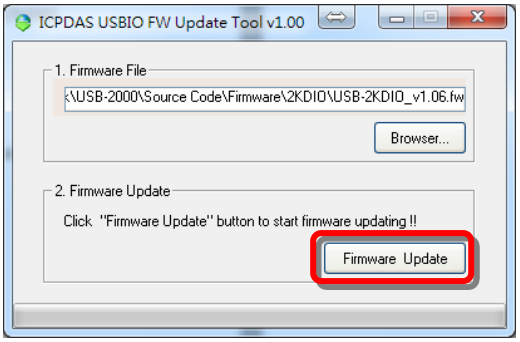


Once the software opened, users can select the firmware file by clicking the "Browser..." button.

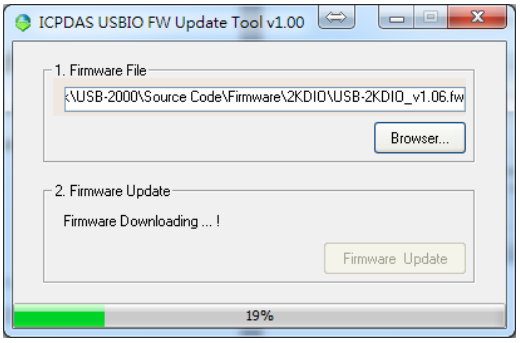

ICPDAS USBIO FW Update Tool v1.00
1. Firmware File
Browser...
2. Firmware Update
Click "Firmware Update" button to start firmware updating !!
Firmware Update

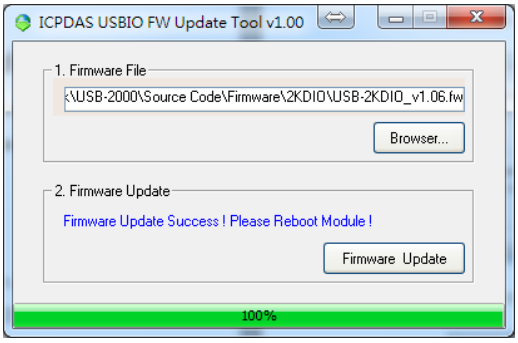
Step 4 Start to download

After selecting correct firmware file for downloading, users can now click the "Firmware Update" button to start to download firmware into the USB I/O module.



Wait for the update message, "Firmware Update Success! Please Reboot Module!", comes up and the download progress to 100%.





Step5 Switch USB I/O module back to RUN mode

After successfully update firmware, users have to switch the mode back to RUN to normal operation. Users also have to use the hardware switch to change the mode.